CR 2007 296
CA029832
528217

SYNTHESIS AND IMPLEMENTATION OF SINGLE AND MULTI-VEHICLE SYSTEMS GUIDANCE BASED ON NONLINEAR CONTROL AND OPTIMIZATION TECHNIQUES

Final Comprehensive Report
(Tasks# 1-14)

(Contract number # W7701-051911)

By:

Dr Brandon. W. Gordon (Supervisor)
Hojjat A. Izadi (PhD Student)
Ali Azimi (PhD Student)
Mehrdad Pakmehr (PhD Student)
Yan Zhao (M A. Sc. Student)
Scientific Authority: Camille-Alain Rabbath

Control and Information Systems (CIS) Lab
Concordia University
Montreal, QC, Canada

March 2006 - March 2007

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **MAR 2007**	2. REPORT TYPE	3. DATES COVERED **00-03-2006 to 00-03-2007**
4. TITLE AND SUBTITLE **Synthesis and Implementation of Single and Multi-Vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Defence R&D Canada Valcartier,2459 Pie-XI Blvd North,Quebec, Quebec,G3J 1X5 Canada,**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**407**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

ABSTRACT

This final report documents the work related to tasks #1 through 14 for the project entitled:

"***Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques***".

with contract number # W7701-051911. The work is conducted by the team of "Control and Information Systems (CIS) Laboratory" of Concordia university during 2006 and 2007. This report includes 8 different chapters to fulfill all the tasks of the mentioned project as follows:

Chapter 1: Tasks #1, 2, 3 (in part) and 5.

Chapter 2: Task # 3 (in part).

Chapter 3: Task # 4.

Chapter 4: Task # 6.

Chapter 5: Task # 7, 8 and 9.

Chapter 6: Task # 10.

Chapter 7: Task # 11.

Chapter 8: Task # 12, 13 and 14.

It is notable that task 3 is accomplished in both chapters 1 and 2.

TABLE OF CONTENTS

CHAPTER 1: HYBRID CONTROL SCHEME AND LPV SLIDING MODE CONTROL FOR DELTA WING ROLL DYNAMICS COUPLED WITH SMA MICRO ACTUATOR DYNAMICS

(Tasks #1, 2, 3 and 5)

Mehrdad Pakmehr
Graduate Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory
Department of Mechanical and Industrial Engineering
Concordia University
Montréal, Québec, Canada H3G 1M8

March 2006

ABSTRACT

This report documents the work related to tasks #1, 2, 3 and #5 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". In this report the radial basis function (RBF) neural network control approach for active flow control extended to handle unmodelled dynamics and multiple equilibria in hybrid (switching) system framework. Hybrid RBF adaptive controller applied to delta wing vortex-coupled roll dynamics using hysteresis switching logic. The combinatory control method also applied to the delta wing dynamics coupled with the SMA micro actuator dynamics which has been obtained form identification process in DRDC. In this report also the linear parameter-varying sliding mode control (LPVSMC) approach which has been developed for linear parameter-varying time-delayed systems (LPVTDS) has been applied to delta wing model coupled with SMA dynamics. This approach combines sliding mode control (SMC), linear parameter-varying (LPV) control theory, and time delay stability analysis to solve a LPVTDS control problem. It is anticipated that this method will lead to significant improvement over existing SMC approaches in aerospace applications with parameter variations and coupled with new SMA actuating devices.

Nomenclature

ρ	air density	I_w	moment of inertia around roll axis
u_∞	free stream velocity	t	observation time
q	dynamic air pressure	T	time period of decay
Cl	roll moment coefficient	T^*	release time
fc	friction coefficient	τ	time at step onset
s_w	wing element area	Γ	circulation or vortex strength
b_w	wing span	Γc	critical circulation
σ	body axis inclination	Λ	effective sweep back angle for left vortex
α	angle of attack	Λ_e	empirical obtained value of sweep back angle
ϕ	roll angle	X_{vb}	non-dimensional vortex breakdown location
λ_0	half apex angle	Xs	static term in X_{vb}

Subscripts

q	quasi-steady term	vb	vortex breakdown
s	static term	l	left vortex
u	unsteady term	r	right vortex
h	time-delayed vector		

Abbreviations

SMA	Shape Memory Alloy
SISO	Single-Input Single-Output
MIMO	Multi-Input Multi-Output
LMI	linear matrix inequality
LPV	linear parameter-varying
LPVSMC	linear parameter-varying sliding mode control
LPVTDS	linear parameter-varying time-delayed system
SMC	sliding mode control
NIRISS	nonlinear indicial response and internal state-space

1.1 Section 1: Introduction, Preliminaries and Motivations

This report addresses the following tasks related to the project "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques":

1) Task 1: Develop robust linear parameter-varying (LPV) based optimal control laws for an existing time-delayed flow control model
2) Task 2: Extend artificial neural network (ANN) control to handle unmodelled dynamics and multiple equilibria
3) Task 3: Integrate outer-loop flow control schemes with inner-loop shape memory alloy actuators
4) Task 5: Build numerical models and simulations through Matlab validating the LPV and ANN flow control approaches

In this report the radial basis function (RBF) neural network control approach for active flow control introduced in references [1,2,4,5] will be extended to handle unmodelled dynamics and multiple equilibria. The theoretical underpinnings of the method are developed in [6, 7, 8].

To deal with the multiple equilibria problem, a multiple model / multiple controller neural network approach will be employed. The controller and neural network will be switched as the system passes closer to another equilibrium point in the model. Appropriate switching surfaces will be employed to achieve this objective. Stability analysis of this hybrid system will be applied to ensure stability and the avoidance of chattering and limit cycles around the switching surfaces.

To handle the nonlinear uncertain model of delta wing roll dynamics we will use hybrid modeling and control notion. Multiple RBF adaptive controllers will be used to control the delta wing along a pre-specified trajectory. We will also use some logic-based switching rules in the control process, and a useful technique to suppress chattering in the switching process.

The combinatory robust adaptive control scheme also will be applied to the delta wing dynamics coupled with the SMA micro actuator dynamics. The SMA model has been obtained from an identification process in DRDC. For SMAs on the left and right sides of the wing two different control loops will be considered.

This report also presents the application of linear parameter-varying sliding mode control (LPVSMC) design approach for a LPVTDS developed in [9] to delta wing model coupled with SMA dynamics. The organization of the report is as follows. The controller synreport conditions are formulated in terms of LMI that can be solved via LMI solvers. This result is then generalized to multiple time-delayed LPV systems. The quasi-LPV modelling and LPVSMC synreport procedure developed [9] for the delta wing roll motion problem will be used for LPVSMC control.

1.1.1 References

[1] M. Pakmehr, B. W. Gordon, C. A. Rabbath, "Robust Adaptive Tracking Control of Delta Wing Vortex-Coupled Roll Dynamics Subject to Delay and SMA Actuator Dynamics", accepted to be presented at the 2005 IEEE International Conference on Mechatronics and Automation, Niagara Falls, Ontario, Canada, July 29th to August 1st 2005.

[2] M. Pakmehr, B. W. Gordon, C. A. Rabbath, "Robust Adaptive Control of Delta Wing Vortex-Coupled Roll Dynamics Using RBF Neural Networks", Proceedings of the 2005 IEEE Conference on Control Applications (CCA05), Toronto, Canada, Aug. 2005, pp 1039-1043.

[3] Pakmehr, M., Gordon, B. W., Rabbath, C. A., "Control-Oriented Modeling and Identification of Delta Wing Vortex-Coupled Roll Dynamics", American Control Conference (ACC) 2005, Portland, Oregon, June 8-10, 2005, pp. 1521-1526.

[4] M. Pakmehr, B. W. Gordon, C. A. Rabbath, "Adaptive Tracking Control of Delta Wing Vortex-Coupled Roll Dynamics Subject to Delay", to be submitted to the AIAA Journal of Guidance, Control and Dynamics, 2006.

[5] M. Pakmehr, B. W. Gordon, C. A. Rabbath, "Robust Stabilizing Control of Delta Wing Vortex-Coupled Roll Dynamics Subject to Delay", to be submitted to the AIAA Journal of Aircraft, 2006.

[6] Slotine, J. -J. E., Li, W., *Applied Nonlinear Control*, Prentice Hall, Englewood Cliffs, New jersey, 1991.

[7] Slotine, J. -J. E., Coetsee, J. A., "Adaptive Sliding Controller Synthesis for Nonlinear Systems", International Journal of Control, Vol. 43, No. 6, 1986, pp 1631-1651.

[8] Sanner, R. M., Slotine, J. -J. E., "Gaussian Networks for Direct Adaptive Control", IEEE Transactions on Neural Networks, Vol. 3, No. 6, Nov. 1992, pp 837-863.

[9] Yang, M., Gordon, B. W., "Linear Parameter-Varying Sliding Mode Control of State Delayed Systems with Application to Delta Wing Vortex Coupled Dynamics", Internal Report (No. 4), submitted to Dr. C. A. Rabbath, Defence Research and Development Canada (DRDC), Valcartier, Quebec, Canada, Nov. 2005.

1.2 <u>Section 2:</u> Hybrid Adaptive Control of Delta Wing Vortex-Coupled Roll Dynamics Subject to Delay

1.2.1 Introduction

From a conceptual point of view the most basic definition of a hybrid system is to immediately specify its behaviour, that is, the set of all possible trajectories of the continuous and discrete variable associated with the system. The "working definition" of the hybrid systems, called the "hybrids automaton model", will provide the framework and terminology to discuss the hybrid systems [1]. The focus on hybrid systems has increased significantly in recent years due to its capability of modelling wide ranges of engineering applications such as:

- Aircraft and air traffic control systems
- Manufacturing systems
- Traffic control systems
- Hierarchical control in process industry
- Electrical networks (circuits with diodes and switches)
- Mechanical systems with collision
- Embedded computation systems

The hybrid framework is ideal for modelling nonlinear physical systems, which include phenomena that occurs at multiple time scales. In these types of systems, the fast dynamics can be abstracted away and be treated as discrete changes affecting the slower dynamic [2].

A hybrid system is a mixture of continuous-time and discrete-event dynamics [2, 3,4]. Therefore, the formulation of equations of the hybrid system motion consists of two main parts:

- Condition or event rules (for discrete event component)
- Evolution of continuous states in every possible modes

By combining differential equations and finite state automata, it is possible to describe a hybrid system. It is important to note that different set of differential equations describe the continuous dynamics in each hybrid systems mode. It means that dynamic structure of a system changes over the time in accordance with its event rules. In more details, the number of state variables and algebraic variables changes when switching among different modes of operation occurs. These systems are denoted as an evolvable structure. Frame works such as dynamic structure discrete event system specification and object oriented physical modelling (OOPM) are introduced for the representation of the systems with evolvable structure [5]. More recently, a hybrid system is developed to describe hybrid dynamics systems of variable structure with varying number of state and algebraic variables [6].

It is worthy to mention that a continuous dynamic in each mode can be described in different form of differential equations such as implicit/explicit Ordinary Differential Equations (ODE) and implicit/explicit Differential Algebraic Equations (DAE) [7].

The preliminary hybrid automaton of delta wing system is shown in Fig. 1:

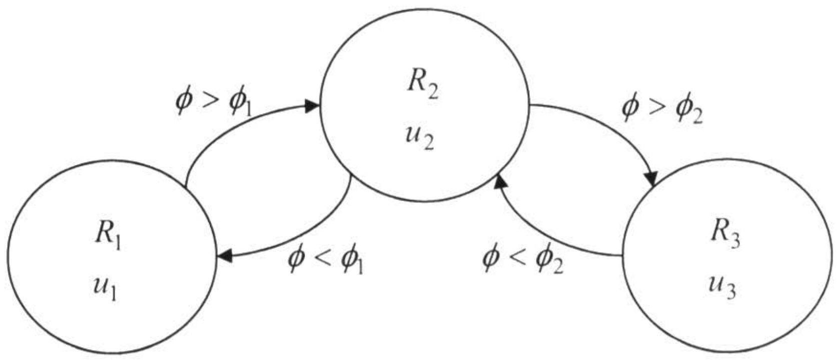


Fig. 1. Hybrid automaton of delta wing system

One of the issues which we will encounter in the defining of the switching logic in hybrid control design process is "*chattering*", which is not desirable. There are basically two ways to suppress chattering: one is called "*hysteresis switching logic*" and the other "*dwell-time switching logic*" [1].

1.2.2 Tracking Controller Structure

In order to further improve performance when large parametric uncertainties are present, we now design an adaptive sliding tracking controller, where rolling moment coefficient (Cl) as the main uncertain parameter is estimated on-line. An important feature of the methodology, however is that it provides a consistent rule for ceasing adaptation. The sliding controller is designed as if the parameters to be estimated were known exactly, i.e. as if adaptation were successfully and exactly complete. Since this is initially no the case, the system first wanders outside the boundary layer, and the information thus generated is used to improve the parameters estimates. Conversely, the scheme recognizes that once in the boundary layer no advantage is gained by further adaptation, since even if the parameters of concern were exactly known, no improvement in tracking performance could be guaranteed formally.

1.2.3 Preliminaries

We try to develop an adaptive tracking control algorithm for dynamic system presented in Section 2. For this system *u(t)* is the control input, Cl is an unknown non-linear function. The control objective is to force the plant state vector, $X = [x_3, x_4]^T$ assumed available for measurement to follow a specified desired trajectory $X_d = [x_d, \dot{x}_d]^T$. Defining the tracking error vector $\tilde{X}(t) = X(t) - X_d(t)$ the problem is thus to design a control law u(t) which ensures that $\tilde{X}(t) \to 0$ as $t \to \infty$.

Reference model which has been applied to the controller is as follows:

$$\begin{cases} x_d(t) = x_d(t_0) + \sum_{i=1}^{M} A_i \sin(k_i t) \\ \dot{x}_d(t) = \sum_{i=1}^{M} A_i k_i \cos(k_i t) \\ \ddot{x}_d(t) = \sum_{i=1}^{M} - A_i k_i^2 \sin(k_i t) \end{cases} \tag{1}$$

$$A_i = A, \; k_i = i, \; i = 1,2,\dots, M$$

Seeking agility and high manoeuvrability capabilities of delta wing in high angle of attack condition, a sophisticated reference trajectory has been selected to show the controller capability to control high performance delta wing.

Using the approaches presented in Ref. [9] and [10], the proposed robust adaptive tracking controller structure is as follows:

$$u_{Track}(t) = u_{pd} + u_{sl} + u_{ad} \tag{2}$$

The components of the proposed controllers are as follows:

A useful tracking error metric for both sliding and adaptive control subsystems is defined by

$$s(t) = \dot{\tilde{x}}(t) + \lambda \tilde{x}(t) \tag{3}$$

where $\lambda > 0$. The equation $s(t) = 0$ defines a time-varying hyper-plane in R^2 on which the tracking error vector decays exponentially to zero [9]. If the magnitude of s can be shown to be bounded by a constant ε (boundary layer), the actual tracking errors can be shown [9] to be asymptotically bounded by:

$$\left|\tilde{x}^{(i)}(t)\right| \le 2^i \lambda^{i-n+1} \varepsilon, \quad i = 0,1,\dots,n-1 \tag{4}$$

1.2.4 Tracking Controller Subsystems

Using the metric $s(t)$ and the saturation function, the control subsystems defined as follows:

$$a_r(t) = [0\ \lambda]\begin{bmatrix} \tilde{x} \\ \dot{\tilde{x}} \end{bmatrix} - \ddot{x}_d \tag{5}$$

$$u_{pd} = -(k_d s(t) + a_r(t)) \tag{6}$$

$$u_{sl} = -k_{sl}\, sat(s(t)/\varepsilon) \tag{7}$$

$$\aleph = [X_{vbl}(t), X_{vbr}(t)]$$

$$u_{ad}(t) = \hat{C}_{l\,A}(\aleph) = (\sum_{i=1}^{N} \hat{w}_i(t)\,\Delta g_i)qs_w b_w \tag{8}$$

$$s_\Delta(t) = s(t) - \varepsilon\ sat(s(t)/\varepsilon) \tag{9}$$

Adaptation law:

$$\dot{\hat{w}}_i = -k_a s_\Delta \Delta g(X_{vbl\&r}(X(t)), \xi_{l\&r_i}) \tag{10}$$

$$\Delta g(X_{vbl\&r}(X(t)), \xi_{l\&r_I}) = Y_{1*N} \tag{11}$$

k_d, k_{sl} and k_a are positive constants which should be chosen for each of the control subsystems.

The online Cl estimation, implemented by gaussian radial basis function (RBF) Neural Networks specially developed for our case using left and right vortex breakdown positions, can be represented mathematically as:

$$\hat{C}_l(\aleph) = (\sum_{i=1}^{N} w_i\,\Delta g_i(X_{vbl\&r}, \xi_{l\&r_i}))qs_w b_w \tag{12}$$

$$\begin{gathered} Xvb_l = (a_l, b_l) \subset R\ \&\ Xvb_r = (a_r, b_r) \subset R, \\ (inputs\ to\ the\ network) \end{gathered} \tag{13}$$

$$\xi_{l_i} = a_l + i\Delta_l,\ \Delta_l = (b_l - a_l)/N,\ \xi_{r_i} = a_r + i\Delta_r,\ \Delta_r = (b_r - a_r)/N \tag{14}$$

$$g_{l_i} = \exp(-(Xvb_l - \xi_{li})/(2\sigma_l^2)), \quad g_{r_i} = \exp(-(Xvb_r - \xi_{ri})/(2\sigma_r^2))$$

$$\Delta g_i = g_{l_i} - g_{ri} \tag{15}$$

$$\sigma_l = (b_l - a_l)/\sqrt{2N^2}, \quad \sigma_r = (b_r - a_r)/\sqrt{2N^2} \tag{16}$$

where g_{li} and g_{ri} are the nonlinear functions implemented by node i in two subcomponents of the first hidden layer and Δg_i is a linear combination of two subcomponents of the first hidden layer in second hidden layer; ξ_{li} and ξ_{ri} represent the input weights (or "center" in the radial basis function literature) of node i, and w_i represents the output weight for that node which is updated using the adaptation law.

Figure 2 shows conceptual illustration of RBF NN used for online estimation of uncertainty, which is used in adaptive control component of tracking controller. This figure shows the two hidden layers of the network, one is combined of two sub-layers and the second hidden layer is a linear combination of these two sub-layers.

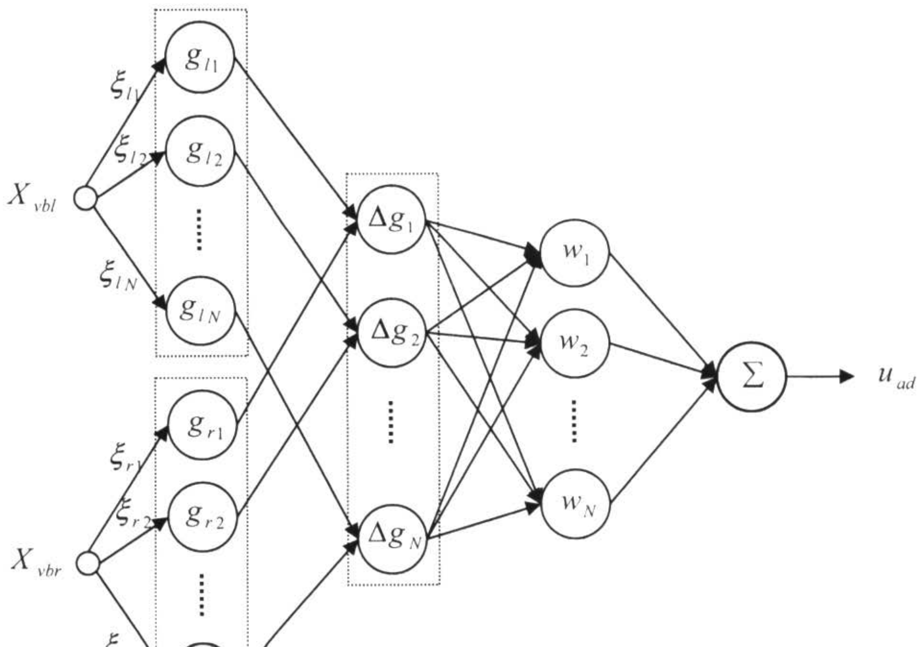


Fig. 2. Radial Basis Function (RBF) neural network structure proposed for online estimation of rolling moment coefficient.

1.2.5 Tracking Controller Stability

To prove the stability issue of our controller the following Lyapunov function candidate has been proposed [9]:

$$V_{track} = \frac{1}{2}(s_\Delta^{2} + \frac{1}{k_a}\sum_{i=1}^{N}\tilde{w}_i^{2}) \tag{17}$$

where $\tilde{w}_i(t) = \hat{w}_i(t) - w_i$. Also the derivative is as follows:

$$\dot{V}_{track} = -(k_d s_\Delta^{2} + |s_\Delta| k_d \varepsilon) - s_\Delta C_l + k_a^{-1}\sum_{i=1}^{N}\tilde{w}_i\dot{\hat{w}}_i \tag{18}$$

It can be shown that with some considerations, $\dot{V} \le -k_d s_\Delta^{2}$, for all $t > t_0$, and if s_Δ and the entire $\tilde{w}_i$ are bounded at time t=0, they remain bounded for all $t > t_0$, the uniform boundedness of $s_\Delta(t)$ by using Barbalat's lemma [8] and this establishes convergence of s_Δ to zero. Hence the inequality $s(t) \le \varepsilon$ holds asymptotically, and the asymptotic bounds on the individual tracking errors follow using (26).

1.2.6 Hybrid Controller Structure

The conceptual structure of hybrid RBF adaptive controller is as follows. Fig. 3 is the illustration of the structure of hybrid adaptive tracking controller.

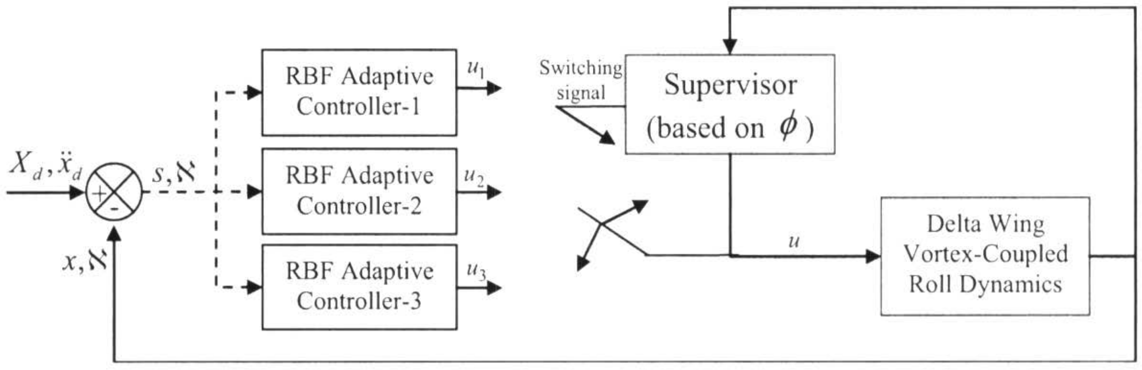

$X_d, \ddot{x}_d$
$s, \aleph$
$x, \aleph$
RBF Adaptive Controller-1
RBF Adaptive Controller-2
RBF Adaptive Controller-3
u_1
u_2
u_3
Switching signal
Supervisor (based on ϕ)
u
Delta Wing Vortex-Coupled Roll Dynamics

Fig. 3. Illustration of the hybrid control structure.

The detailed hybrid automaton for delta wing hybrid control case has been illustrated in Fig. 4.

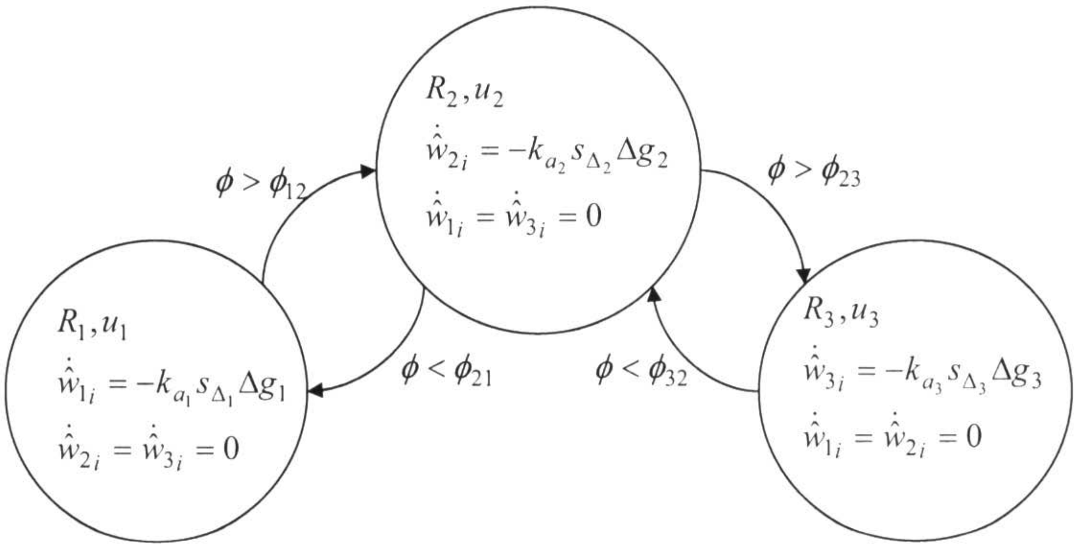

R_2, u_2
$\dot{\hat{w}}_{2_i} = -k_{a_2} s_{\Delta_2} \Delta g_2$
$\dot{\hat{w}}_{1_i} = \dot{\hat{w}}_{3_i} = 0$
R_1, u_1
$\dot{\hat{w}}_{1_i} = -k_{a_1} s_{\Delta_1} \Delta g_1$
$\dot{\hat{w}}_{2_i} = \dot{\hat{w}}_{3_i} = 0$
R_3, u_3
$\dot{\hat{w}}_{3_i} = -k_{a_3} s_{\Delta_3} \Delta g_3$
$\dot{\hat{w}}_{1_i} = \dot{\hat{w}}_{2_i} = 0$
$\phi > \phi_{12}$
$\phi > \phi_{23}$
$\phi < \phi_{21}$
$\phi < \phi_{32}$

Fig. 4. Detailed Hybrid automaton of the switching controller

The supervisor in Fig. 3 is called switching logic. One of the main problems in the design of a switching logic is that usually it is not desirable to have "chattering", that is, very fast switching. There are basically two ways to suppress chattering: one is called "hysteresis switching logic" and the other is "dwell-time switching logic". Fig. 5 shows the hysteresis switching logic which has been applied to the hybrid control of the delta wing system.

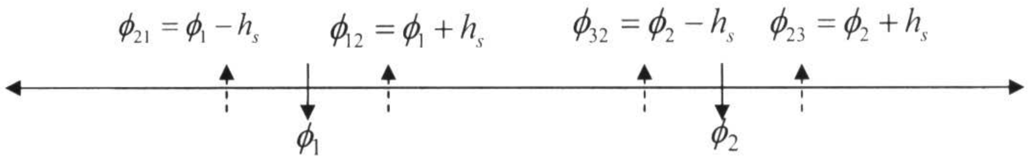

$\phi_{21} = \phi_1 - h_s$
$\phi_{12} = \phi_1 + h_s$
$\phi_{32} = \phi_2 - h_s$
$\phi_{23} = \phi_2 + h_s$
ϕ_1
ϕ_2

Fig. 5. Hysteresis switching logic for hybrid control of delta wing

This hysteresis switching logic is a way for smooth switching between different models and controllers.

1.2.7 Numerical Simulation and Discussion

The initial conditions for all of these simulations set as: $v(\theta): x_1(\theta) = x_2(\theta) = 0, x_3(\theta) = 10\,[\text{deg}], x_4(\theta) = 100[\text{deg/sec}]$. The complexity of the reference trajectory has been defined by setting the related parameters to be $A = 0.05, M = 6$. The switching logic parameters are set as follows: $\phi_1 = -20\,[\text{deg}], \phi_2 = 20\,[\text{deg}], h_s = 4[\text{deg}]$. The parameters for the *tracking controller* of the *first region*, have been chosen as $k_{d1} = 1, k_{sl1} = 1, k_{a1} = 20,\ \lambda_1 = 30, N_1 = 20$ and $\varepsilon_1 = 0.1$, for the *second region* $k_{d2} = 1, k_{sl2} = 1, k_{a2} = 15,\ \lambda_2 = 20, N_2 = 20$ and $\varepsilon_2 = 0.1$ and for the *third region* $k_{d3} = 1, k_{sl3} = 1, k_{a3} = 10,\ \lambda_3 = 10, N_3 = 20$ and $\varepsilon_3 = 0.1$. The simulation results for the proposed controller are shown in Fig. 6 to Fig. 15. Figure 6, shows the time history of the $x_3(t)$ state and its reference; tracking performance is acceptable. Figure 7, shows the time history of $x_4(t)$ state and its reference. Figure 8, shows the time history of $x_1(t)$ and $x_2(t)$ states. Figure 9 and 10, show error and error derivative time histories. Figure 11, shows the control input time history; chattering in the control input is due to the switching task, it has been optimized by adjusting h_s. Figure 12, shows the control input components time history. Figure 13 show the sliding time history; as it is apparent variation is inside the proposed boundary layer but there are some big jumps. Figure 14, shows the time history of the left vortex breakdown location on the wing. Figure 15, shows the time history of the right vortex breakdown location on the wing. Figures show that the controller is stable and the tracking performance is good enough.

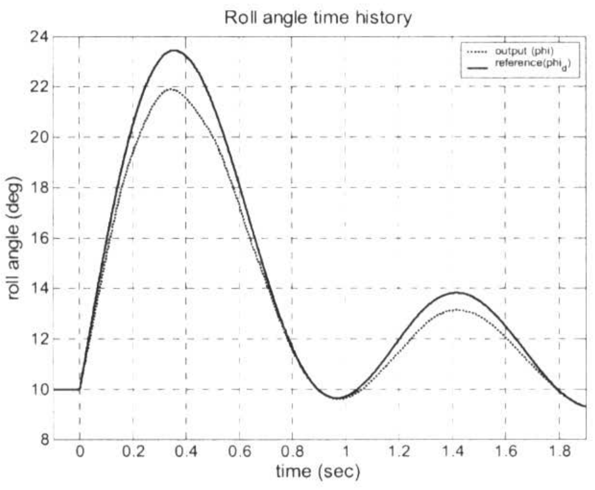


Fig. 6 ϕ and ϕ_d time history (plant output and the command signal)

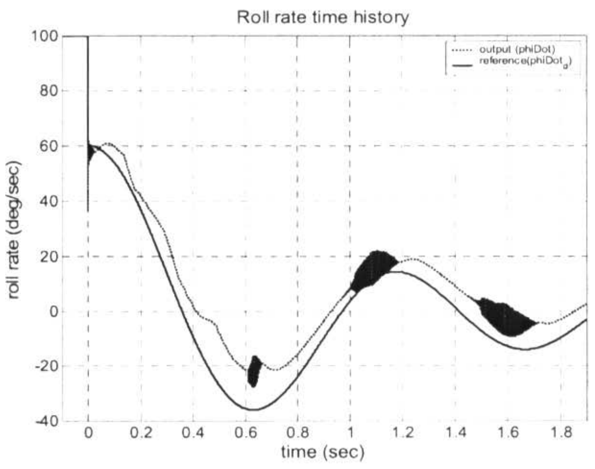


Fig. 7 $\dot{\phi}$ and $\dot{\phi}_d$ time history

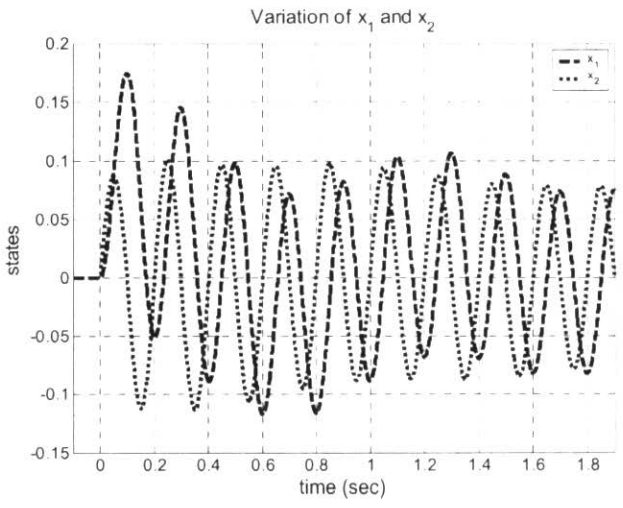


Fig. 8 x_1 and x_2 time history

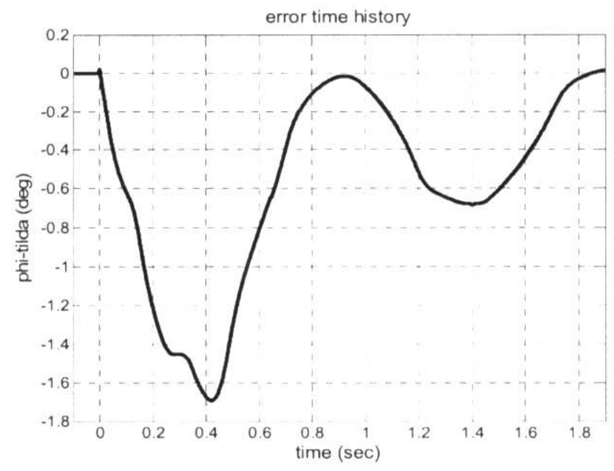


Fig. 9 error ($\tilde{\phi} = \phi - \phi_d$) time history

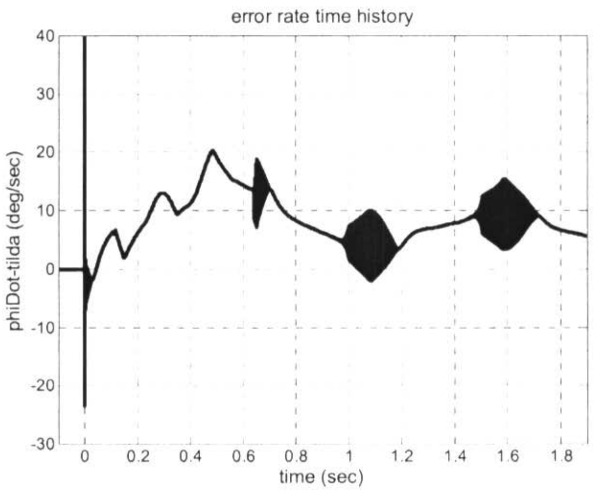


Fig. 10 $\dot{\tilde{\phi}} = \dot{\phi} - \dot{\phi}_d$ time history

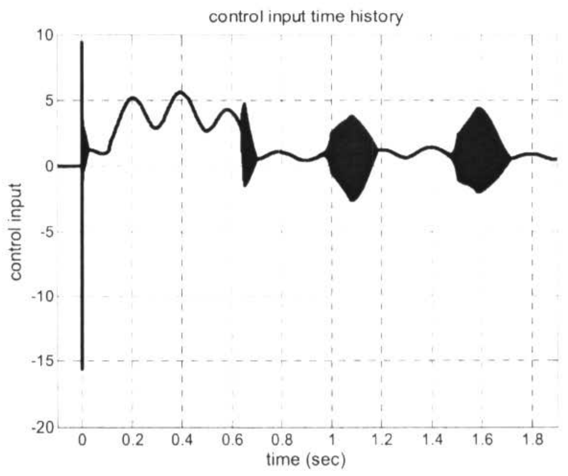


Fig. 11 u (overall control input) time history

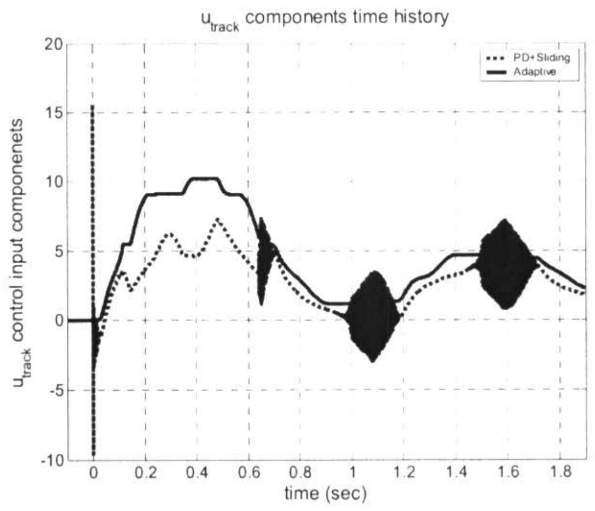


Fig. 12 u control input components time history

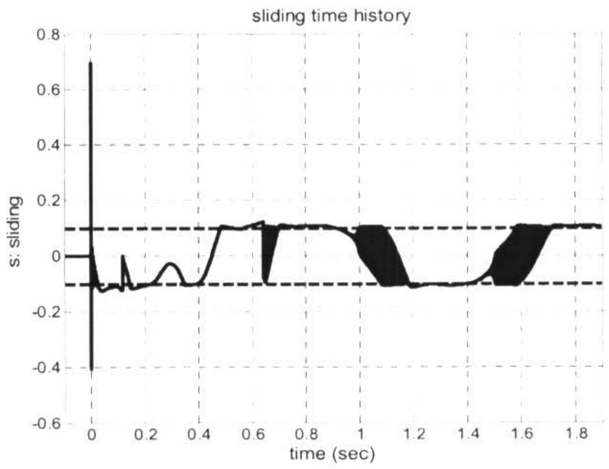


Fig. 13 $s = \dot{\tilde{\phi}} - \lambda\tilde{\phi}$ (sliding) time history

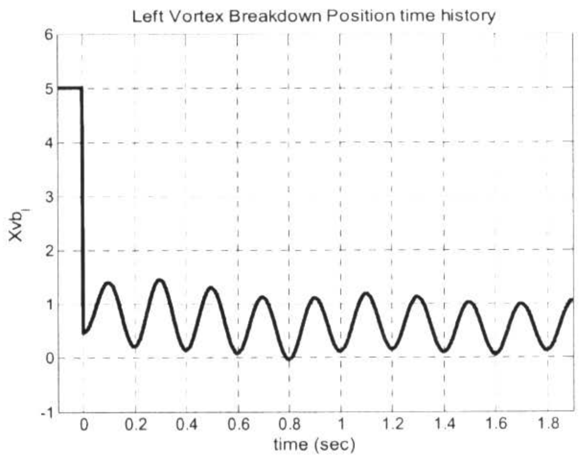


Fig. 14 Xvb_l time history

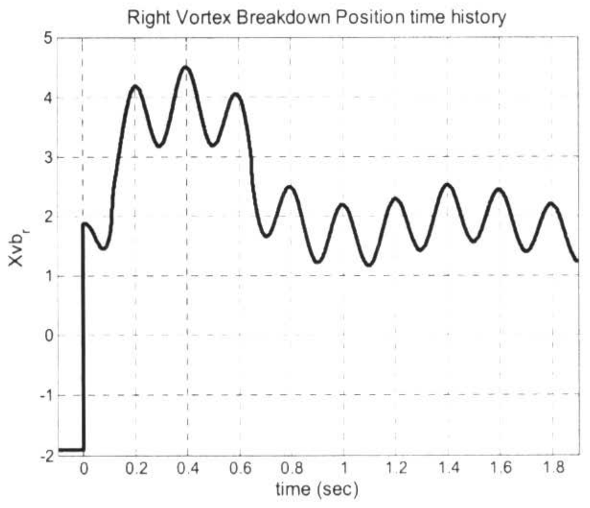


Fig. 15 Xvb_r time history

1.2.8 References

[1] Schaft, A., Schumacher H., *An introduction to Hybrid Dynamical Systems* (Lecture notes in control and information sciences 251), Springer, London, GB, 2000.

[2] Mosterman, P. J., Zhao F., and Biswas, G., "Model Semantics and Simulation for Hybrid Systems Operating in Sliding Regimes," *Proceedings of the AAAI Fall Symposium 97 on Model-directed Autonomous Systems*, Boston, MA, 1997, pp. 48-55.

[3] Hespanha, J. P., "Lecture notes on hybrid control and switched systems", http://www.ece.ucsb.edu/~hespanha/ece229/, University of California at Santa Barbara, Fall 2005.

[4] Johansson, K. H., Lygeros, J., Sastry. S., "Modelling of Hybrid systems," In H. Unbehauen, Ed., Encyclopedia of Life Support Systems (EOLSS), Theme 6.43: Control Systems, Robotics and Automation. Developed under the auspices of UNESCO, 2004. Article-level contribution. Invited paper.

[5] Barros, F. J., Zeigler B. P., and Fishwick, P. A., "Multimodel and Dynamic Structure Models: An Integration of DSDE/DEVS and OOPM," *Winter Simulation Conference*, Washington DC, 1998, pp.413-419.

[6] Urquia, A., Dormido, S., "Object-Oriented Description of Hybrid Dynamic

Systems of Variable Structure," SAGE J. Simulation, 79(9), 2003, 485-493.

[7] Brenan, K., Campbell, S., Petzold, L., *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, Notrh-Holland, Amsterdam, 1989.

[8] Slotine, J. -J. E., Li, W., *Applied Nonlinear Control*, Prentice Hall, Englewood Cliffs, New jersey, 1991.

[9] Slotine, J. -J. E., Coetsee, J. A., "Adaptive Sliding Controller Synthesis for Nonlinear Systems", *International Journal of Control*, Vol. 43, No. 6, 1986, pp 1631-1651.

[10] Sanner, R. M., Slotine, J. -J. E., "Gaussian Networks for Direct Adaptive Control", *IEEE Transactions on Neural Networks*, Vol. 3, No. 6, Nov. 1992, pp 837-863.

1.3 Section 3: Adaptive Control of Delta Wing Roll Dynamics Coupled with SMA Micro-Actuator Dynamics

In this Section the combinatory control scheme developed in [1] will be applied to the model coupled with the new SMA model developed in [2]. The SMA control loop has three main blocks: 1) low pass filter; 2) control law with an integrator; 3) SMA model (from identification process).

To integrate the SMA control block in our delta wing vortex-coupled dynamics, the discrete time model in equation (19) has been changed to the continuous time model presented in equation (20) using Euler method.

$$\begin{aligned} x(k+1) &= A_{dt}x(k) + B_{dt}u(k) \\ y(k) &= C_{dt}x(k) + D_{dt}u(k) \end{aligned} \tag{19}$$

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t) + D_c u(t) \end{aligned} \tag{20}$$

Applying the Euler method for numerical integration the following holds

$$x(k+1) = x(k) + \dot{x}T_s \tag{21}$$

Replacing $\dot{x}$, the following holds

$$x(k+1) = x(k) + (A_c x(k) + B_c u(k))T_s \tag{22}$$

Replacing $x(k+1)$, the following holds

$$A_{dt}\, x(k) + B_{dt}\, u(k) = (I + A_c T_s)x(k) + B_c\, T_s\, u(k) \tag{23}$$

$$\begin{aligned} A_{dt} &= I + A_c T_s \\ B_{dt} &= B_c T_s \end{aligned} \tag{24}$$

So the matrices in continues time form will be computed as follows

$$\begin{aligned} A_c &= (A_{dt} - I) T_s^{-1} \\ B_c &= B_{dt} T_s^{-1} \end{aligned} \tag{25}$$

C and D will remain unchanged

$$\begin{aligned} C_c &= C_{dt} \\ D_c &= D_{dt} \end{aligned} \tag{26}$$

The Simulink block diagram of the *discrete time* model of SMA control loop is shown in Figure 1.

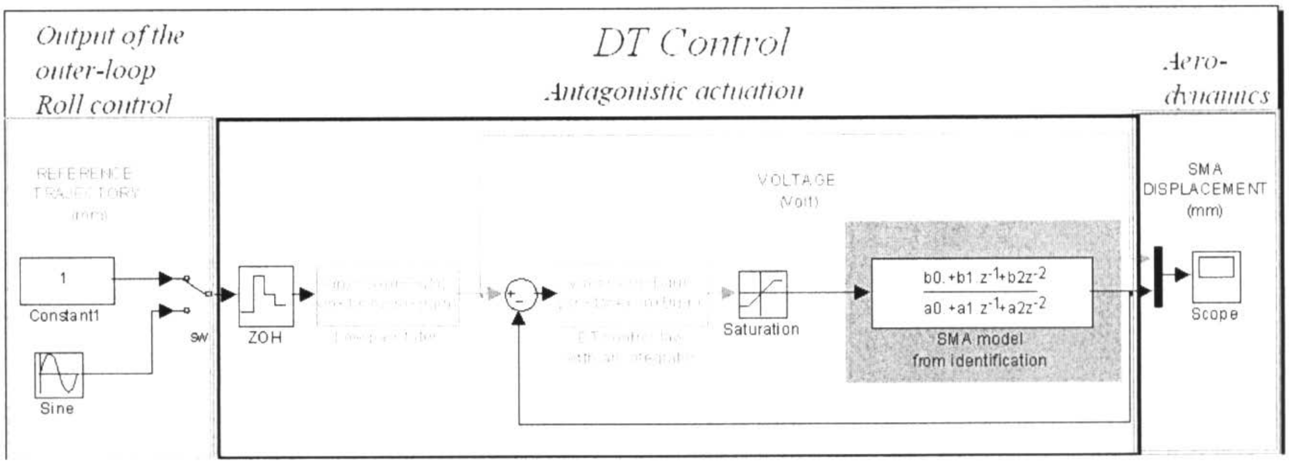

Output of the outer-loop Roll control
DT Control
Antagonistic actuation
Aero-dynamics
REFERENCE TRAJECTORY (mm)
1
Constant1
Sine
SW
ZOH
VOLTAGE (Volt)
Saturation
b0.+b1 z-1+b2z-2 / a0 +a1 z-1+a2z-2
SMA model from identification
SMA DISPLACEMENT (mm)
Scope

Fig. 1 Simulink block diagram of the *discrete time* model of SMA control loop is shown

The Simulink block diagram of the *continuous time* model of SMA control loop is shown in Figure 2.

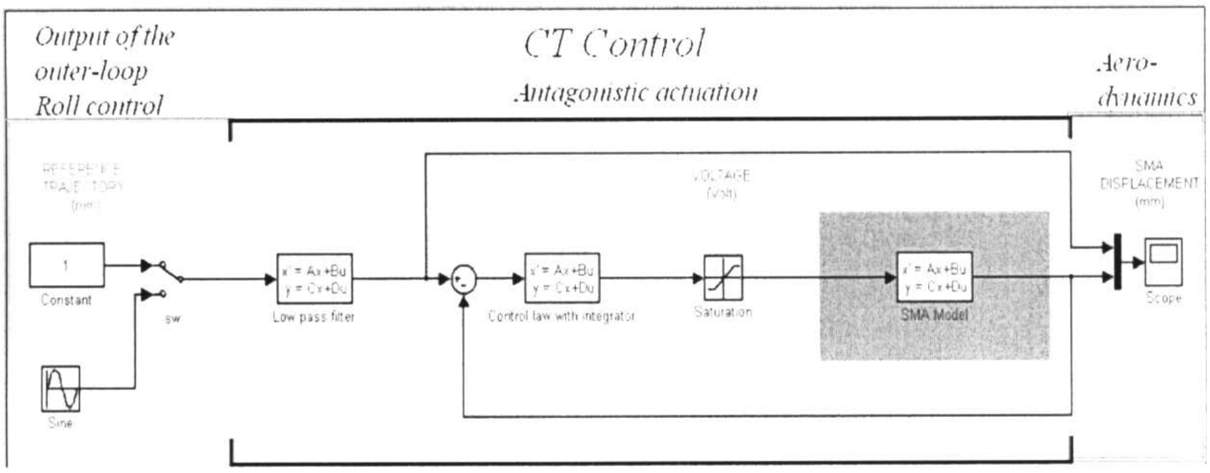


Fig. 2 Simulink block diagram of the *continuous time* model of SMA control loop

Figure 3 shows a plot which is the comparison between the original discrete time model and the continuous time model we obtained.

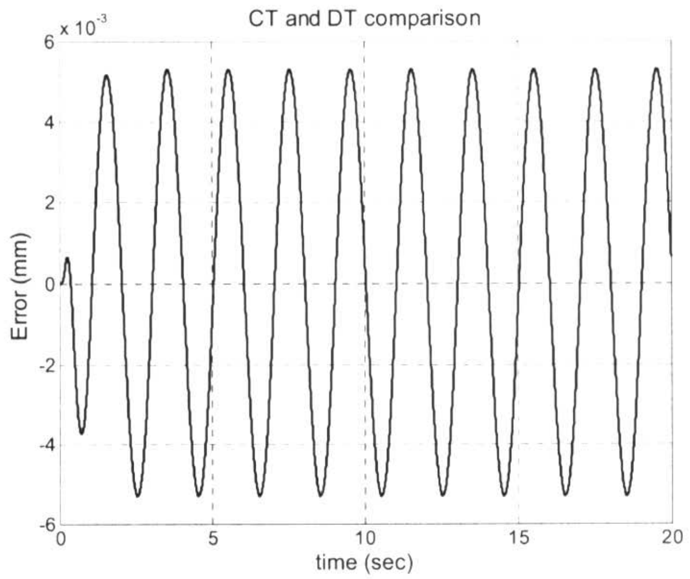


Fig. 3 Comparison between the discrete and continuous time SMA control loop simulation

As Figure 3 shows, the obtained continuous time model difference from the original discrete time model is in an acceptable range, and it can be used in the simulations.

1.3.1 Delta Wing Dynamics Coupled with SMA Antagonistic Actuation System

The delta wing vortex coupled model will be as follows:

$$\begin{cases} \dot{x}_1(t) = c\,x_2(t) \\ \dot{x}_2(t) = -c\,x_1(t) - \varepsilon_d\,x_2(t) + x_4(t) + x_4(t-T) \\ \dot{x}_3(t) = -\varepsilon_d\,x_3(t) + x_4(t) \\ \dot{x}_4(t) = -C_l(\aleph)Q - f_c\,x_4(t) + h_1 u_1(t)/I_w \end{cases} \tag{27}$$

Where

$$\aleph = [X_{vbl}(t), X_{vbr}(t)] \tag{28}$$

$$X_{vbl}(t) = X_{sl}(x(t)) + X_{sl}(x(t))k_q(t)x_4(t) + a(t)x_1(t) + y_{smal}(t) \tag{29}$$

$$X_{vbr}(t) = X_{sr}(x(t)) + X_{sr}(x(t))k_q(t)x_4(t) - a(t)x_1(t) + y_{smar}(t) \tag{30}$$

The conceptual flowchart of the delta wing vortex-coupled dynamics with the new SMA model is shown in Figure 4.

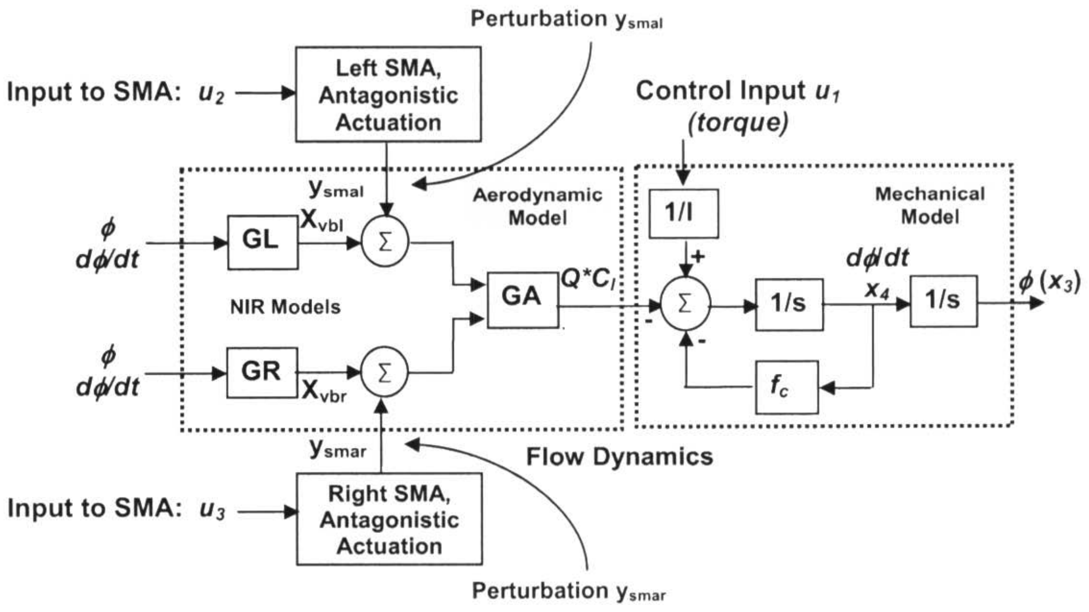


Fig. 4 Conceptual flowchart of the delta wing vortex-coupled dynamics with the new SMA model

The SMA control loop contains the continuous time model of the following items:

1- Low pass filter:

$$\begin{aligned} \dot{x}_{lpf}(t) &= A_{lpf} x_{lpf}(t) + B_{lpf} u_i(t) \\ y_{lpf}(t) &= C_{lpf} x_{lpf}(t) + D_{lpf} u_i(t) \end{aligned} \tag{31}$$

Where

$$u_i(t) = u_{2-0r-3}(t) \tag{32}$$

The numerical values of the matrices for the continuous time model are as follows:

$$A_{lpf} = \begin{bmatrix} -20.0000 & -100.0000 \\ 01.0000 & -000.0000 \end{bmatrix},\ B_{lpf} = \begin{bmatrix} 1.0000 \\ 0.0000 \end{bmatrix},\ C_{lpf} = \begin{bmatrix} 0.0000 & 100.0000 \end{bmatrix},\ D_{lpf} = [0.0000] \tag{33}$$

2- Control law with an integrator:

$$\begin{aligned} \dot{x}_{cont}(t) &= A_{cont} x_{cont}(t) + B_{cont} u_{cont}(t) \\ y_{cont}(t) &= C_{cont} x_{cont}(t) + D_{cont} u_{cont}(t) \end{aligned} \tag{34}$$

Where

$$u_{cont}(t) = (y_{lpf}(t) - y_{sma}(t)) \tag{35}$$

The numerical values of the matrices for the continuous time model are as follows:

$$A_{cont} = \begin{bmatrix} 0.0000 & 108.5991 & -117.8455 \\ -0.0000 & 174.8300 & 226.4447 \\ 0.0000 & -177.2327 & -223.8374 \end{bmatrix},\ B_{cont} = \begin{bmatrix} 27.6270 \\ 1.9049 \\ -2.4398 \end{bmatrix},\ C_{cont} = \begin{bmatrix} 1.0000 & 1.0000 & 0.0000 \end{bmatrix},\ D_{cont} = [0.1433] \tag{36}$$

3- SMA model from system identification process:

$$\begin{aligned}\dot{x}_{sma}(t) &= A_{sma}x_{sma}(t) + B_{sma}u_{sma}(t)\\ y_{sma}(t) &= C_{sma}x_{sma}(t) + D_{sma}u_{sma}(t)\end{aligned} \qquad (37)$$

Where

$$u_{sma}(t) = sat(y_{cont}(t)) \qquad (38)$$

The saturation function has been defines as follows:

$$sat(y_{cont}(t)) = \begin{cases} y_{cont}(t) \geq 6 \rightarrow sat(y_{cont}(t)) = 6 \\ -6 \leq y_{cont}(t) \leq 6 \rightarrow sat(y_{cont}(t)) = y_{cont}(t) \\ y_{cont}(t) \leq -6 \rightarrow sat(y_{cont}(t)) = -6 \end{cases} \qquad (39)$$

The numerical values of the matrices for the continuous time model are as follows:

$$A_{sma} = \begin{bmatrix} 177.3034 & -177.4760 \\ 224.4827 & -224.2960 \end{bmatrix},\ B_{sma} = \begin{bmatrix} 107.8497 \\ -116.6330 \end{bmatrix},\ C_{sma} = \begin{bmatrix} -0.0195 & 0.0210 \end{bmatrix},\ D_{sma} = \begin{bmatrix} 0.0000 \end{bmatrix} \qquad (40)$$

1.3.2 Combinatory Controller Structure

The robust adaptive tracking controller structure which has been developed in [1] is as follows:

$$\begin{aligned} u(t) &= u_{Track}(t) + u_{Stab}(t) \\ &= u_{pd}(t) + u_{sl}(t) + u_{ad}(t) + u_{\rho}(x_t, t) \\ &= -(k_d s(t) + a_r(t)) - k_{sl}\, sat(s(t)/\varepsilon) + (\sum_{i=1}^{N} \hat{w}_i(t)\Delta g_i)qs_w b_w - \mu_0 H_0 y(t) \end{aligned} \qquad (41)$$

The control inputs for the SISO model coupled with SMA actuator dynamics, are as follows:

$$u_1(t) = u(t) \tag{42}$$

$$u_2(t) = \rho_{vbl} u(t) \tag{43}$$

$$u_3(t) = -\rho_{vbr} u(t) \tag{44}$$

ρ_{vbl} and ρ_{vbr} are coefficients for the inputs to the left and right SMA control loop.

The conceptual illustration of the combinatory control structure coupled with the new SMA antagonistic actuation model id shown in Figure 5.

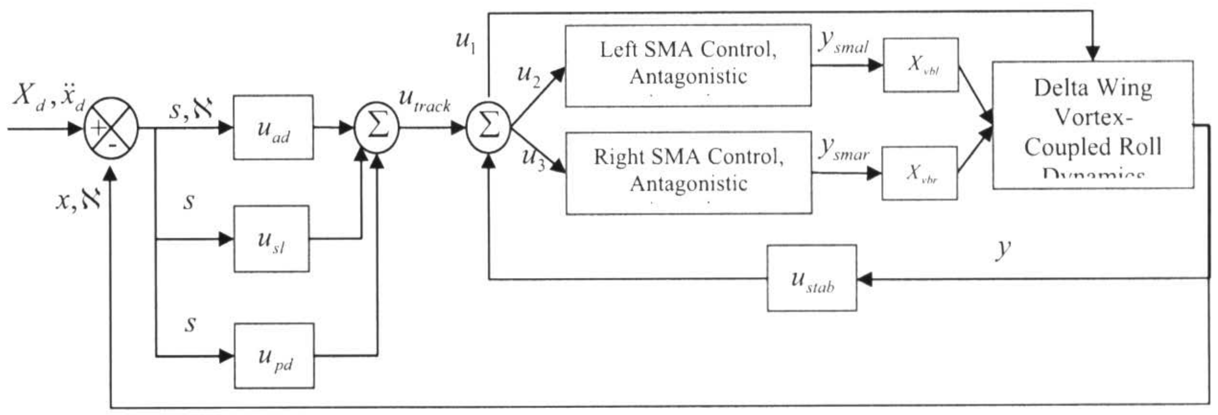


Fig. 5 Conceptual illustration of the combinatory control structure coupled with new SMA model.

1.3.3 Numerical Simulation and Discussion

The initial condition for the simulation set as: $v(\theta): x_1(\theta) = x_2(\theta) = 0, x_3(\theta) = 30$ [deg], $x_4(\theta) = 100$[deg/sec] . The complexity of the reference trajectory has been defined by setting the related parameters to be $A = 0.05$, $M = 6$. The parameters for the *tracking controller* have been chosen as

$k_d = 1, k_{sl} = 1, k_a = 10$, $\lambda = 25, N = 20$ and $\varepsilon = 0.1$. We chose $\eta = T = 0.1[\text{sec}]$ and $\mu_0 = 200$ for the stabilizing control and also we assumed $\rho_{vbl} = \rho_{vbr} = 0.05$. The simulation results for the proposed controller are shown in Figure 6 to Figure 17. Figure 6, shows the time history of the $x_3(t)$ state and its reference; tracking performance is acceptable. Figure 7, shows the time history of $x_4(t)$ state and its reference. Figure 8, shows the time history of $x_1(t)$ and $x_2(t)$ states. Figure 9 and 10, show error and error derivative time histories. Figure 11, shows the control input time history. Figure 12 and 13 show the control input components time history. Figure 14 show the sliding time history; as it is apparent variation is inside the proposed boundary layer but there are some big jumps. Figure 15, shows the time history of the left vortex breakdown location on the wing. Figure 16, shows the time history of the right vortex breakdown location on the wing. Figure 17, shows the output time history of the right and left SMA control loops. Figures show that the controller is stable and the tracking performance is good enough with the perturbations that we add to system by using the SMA micro-actuators control for left and right halves of the wing.

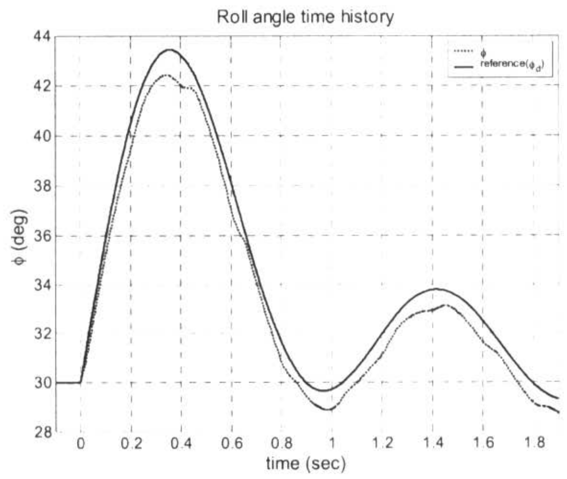


Fig. 6 ϕ and ϕ_d time history (plant output and the command signal)

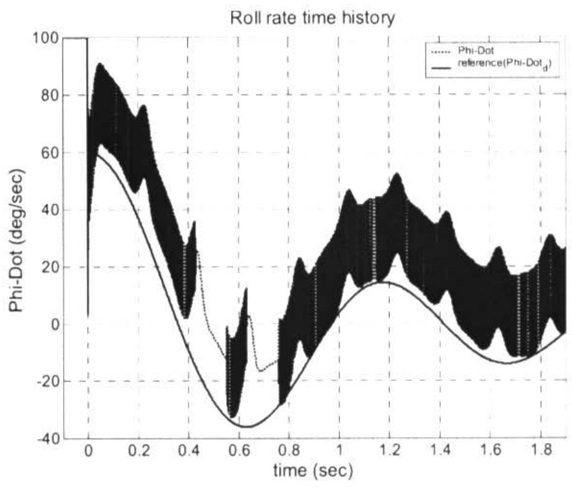


Fig. 7 $\dot{\phi}$ and $\dot{\phi}_d$ time history

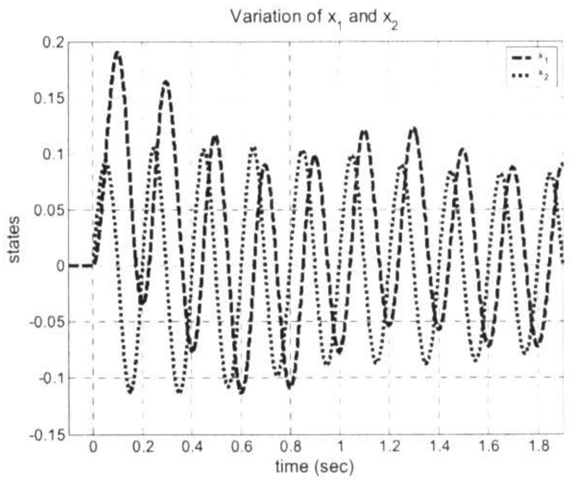


Fig. 8 x_1 and x_2 time history

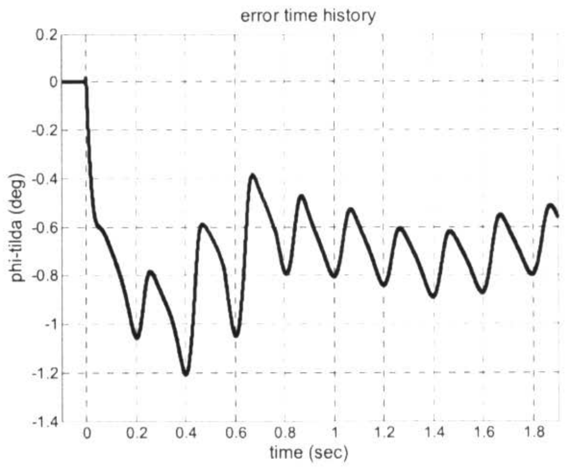


Fig. 9 error ($\tilde{\phi} = \phi - \phi_d$) time history

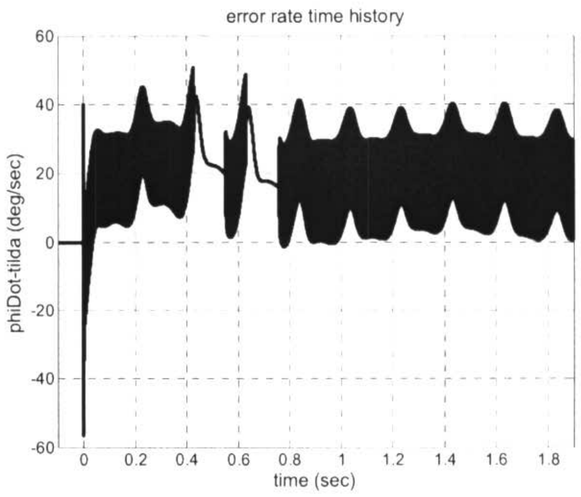


Fig. 10 $\dot{\tilde{\phi}} = \dot{\phi} - \dot{\phi}_d$ time history

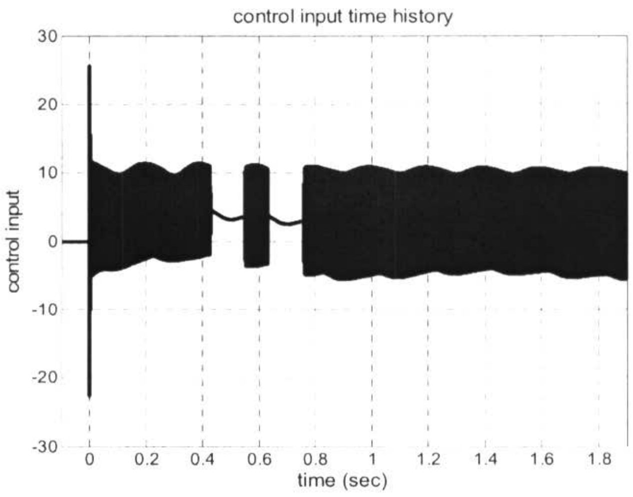


Fig. 11 u_1 (overall control input) time history

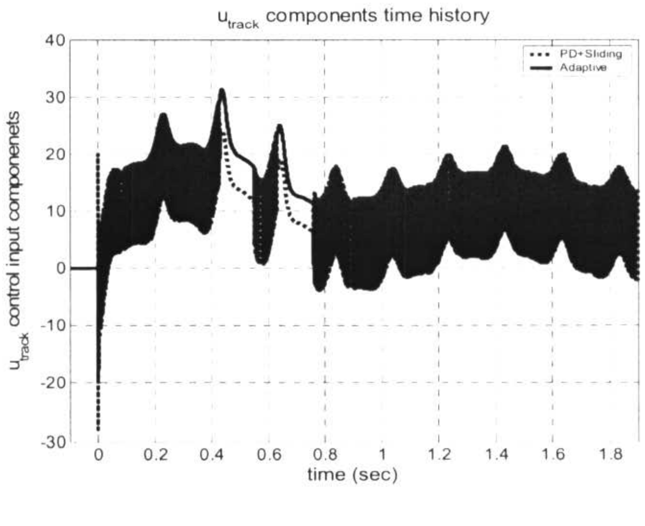


Fig. 12 u_{track} control input components time history

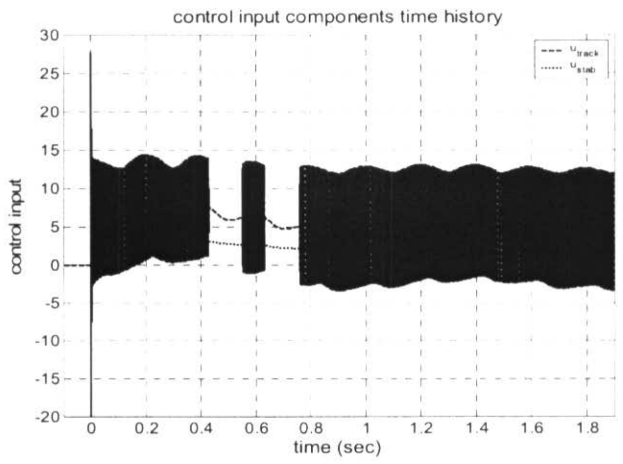


Fig. 13 u_1 control input components time history

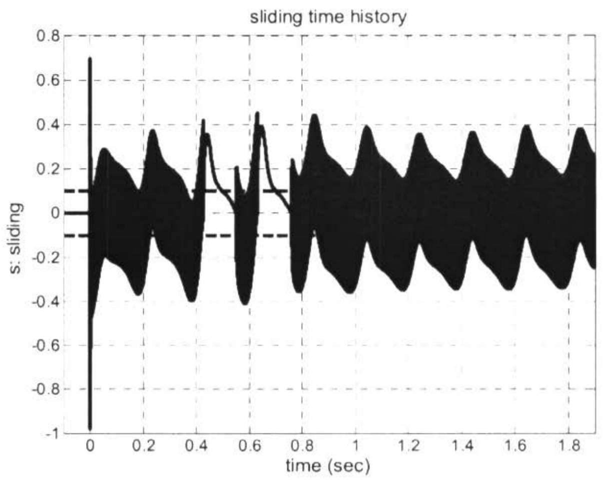


Fig. 14 $s = \dot{\tilde{\phi}} - \lambda\tilde{\phi}$ (sliding) time history

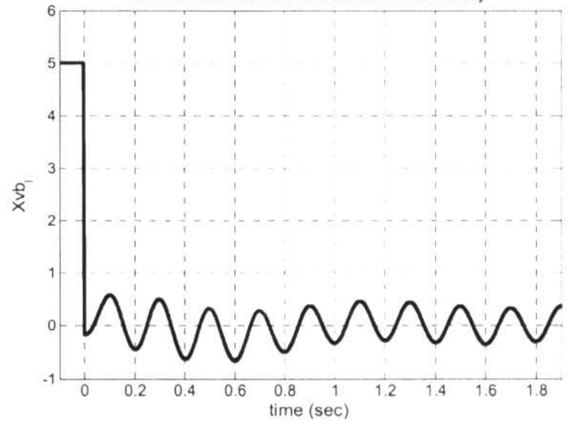


Fig. 15 Xvb_l time history

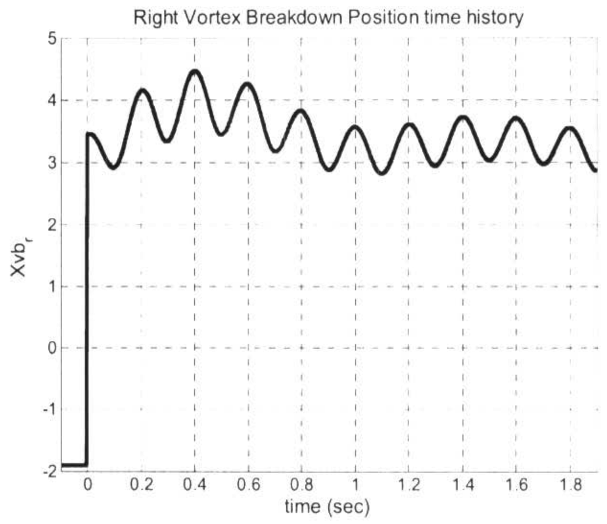


Fig. 16 Xvb_r time history

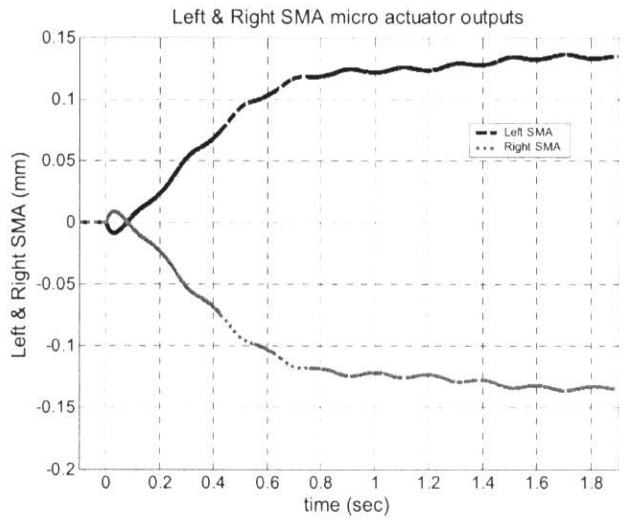


Fig. 17 Left and Right SMA control loop output time history

1.3.4 References

[1] Pakmehr, M., Gordon, B. W., "Final Control Scheme for Delta Wing Roll Dynamics", Internal Report, submitted to Dr. C. A. Rabbath, Defence Research and Development Canada (DRDC), Valcartier, Quebec, Canada, March 2005.

[2] Léchevin, N., Boissoneault, O., and Rabbath, C. A., "Control of Antagonistic Actuator by

Means of

Shape Memory Alloy - Identification-Based Control", Internal Report, Defence Research and Development Canada (DRDC), Valcartier, Quebec, Canada, January 5, 2006.

1.4 Section 4: Application of LPVSMC to Vortex-Coupled Delta Wing Roll Dynamics Coupled with SMA

1.4.1 Preliminaries

Refer to ref [1].

1.4.2 Quasi-LPV Modelling for Delta Wing Systems [1]

In this section, a quasi-LPV model is built on the aerodynamic model of vortex-coupled delta wing systems. At first, the integral of sinusoid in equation (45)

$$\int_{t-T}^{t} X_u(t-\tau)\dot{\Phi}(\tau)d\tau = \frac{1.65}{\tan\alpha(t)}\int_{t-T}^{t}\sin(\frac{\pi(t-\tau)}{T^*})\dot{\Phi}(\tau)d\tau \tag{45}$$

is approximated as a constant in several short intervals. Then, the integral is replaced by the sum of three time-delayed components. Next, a state equation is constructed. In addition, the time-varying parameters are defined and the LPV state equations of the delta wing systems are obtained.

In order to approximate the integral in equation (45), the whole integral range, $t - T \le \tau \le t$, is divided into four equal intervals, i.e. $\begin{bmatrix} t & t\text{-}\frac{T}{4}\end{bmatrix}$, $\begin{bmatrix} t\text{-}\frac{T}{4} & t\text{-}\frac{T}{2}\end{bmatrix}$, $\begin{bmatrix} t\text{-}\frac{T}{2} & t\text{-}\frac{3T}{4}\end{bmatrix}$ and $\begin{bmatrix} t\text{-}\frac{3T}{4} & t\text{-}T\end{bmatrix}$. Since the size of each interval is very small (0.04sec), it is reasonable to consider the $\sin(\frac{\pi(t-\tau)}{T^*})$ term as a constant. Now, we can approximate the whole integral as

$$\begin{aligned}
&\int_{t-T}^{t}\sin\left(\frac{\pi(t-\tau)}{T^{*}}\right)\dot{\Phi}(\tau)d\tau \\
&\approx \frac{\sin\left(\frac{\pi(t-t)}{T}\right)+\sin\left(\frac{\pi(t-(t-\frac{T}{4}))}{T}\right)}{2}\int_{t-\frac{T}{4}}^{t}\dot{\Phi}(\tau)d\tau \\
&+\frac{\sin\left(\frac{\pi(t-(t-\frac{T}{4}))}{T}\right)+\sin\left(\frac{\pi(t-(t-\frac{T}{2}))}{T}\right)}{2}\int_{t-\frac{T}{2}}^{t-\frac{T}{4}}\dot{\Phi}(\tau)d\tau \\
&+\frac{\sin\left(\frac{\pi(t-(t-\frac{T}{2}))}{T}\right)+\sin\left(\frac{\pi(t-(t-\frac{3T}{4}))}{T}\right)}{2}\int_{t-\frac{3T}{4}}^{t-\frac{T}{2}}\dot{\Phi}(\tau)d\tau \\
&+\frac{\sin\left(\frac{\pi(t-(t-\frac{3T}{4}))}{T}\right)+\sin\left(\frac{\pi(t-(t-T))}{T}\right)}{2}\int_{t-T}^{t-\frac{3T}{4}}\dot{\Phi}(\tau)d\tau
\end{aligned} \tag{46}$$

From the Leibniz-Newton formula

$$\int_{t-h}^{t}\dot{\mathbf{x}}(s)ds=\mathbf{x}(t)-\mathbf{x}(t-h) \tag{47}$$

We can obtain the approximation of the integral terms

$$\int_{t-T}^{t}\sin\left(\frac{\pi(t-\tau)}{T^{*}}\right)\dot{\Phi}(\tau)d\tau\approx 0.354\Phi(t)+0.5\Phi(t-T/4)-0.5\Phi(t-3T/4)-0.354\Phi(t-T) \tag{48}$$

For simplification, we define a new variable to represent the sum of these three time delays

$$\frac{1}{2}\delta=\left[0.5\Phi(t-T/4)-0.5\Phi(t-3T/4)-0.354\Phi(t-T)\right] \tag{49}$$

After replacing the integral terms by time-delayed representation, the vortex breakdown location can be represented as

$$X_{vbl}(t)=X_{sl}+\frac{1.65}{\tan\alpha(t)}\left(0.354\Phi(t)+\frac{1}{2}\delta\right) \tag{50}$$

$$X_{vbr}(t)=X_{sr}-\frac{1.65}{\tan\alpha(t)}\left(0.354\Phi(t)+\frac{1}{2}\delta\right) \tag{51}$$

By substituting equations and into equation (52):

$$Cl=e_0+e_1(X_{vbl}-X_{vbr}) \tag{52}$$

The roll moment coefficient $Cl(X_{vbl},X_{vbl})$ can be represented as

$$Cl = e_0 + e_1 \cdot \left[(X_{sl} - X_{sr}) + \frac{1.65}{\tan \alpha(t)} \left(0.708\,\Phi(t) + \delta\right) \right] \tag{53}$$

The LPV modelling approach focuses on linearizing the nonlinear systems by hiding the nonlinear terms via defining them as time-varying parameters. Before we construct the LPV model for the roll motion of delta wing systems, the roll angle of delta wing is defined as

$$x_1 = \Phi(t) \tag{54}$$

and the roll angular velocity of delta wing as

$$x_2 = \dot{\Phi}(t) \tag{55}$$

It is hard to analyze the systems and to synthesize the controller for the steady terms X_{sl} and X_{sr}, since they are the solutions of the second-order equation (56).

$$C_0 + BX_s - AX_s^2 = \Gamma \tag{56}$$

Therefore, precise mathematical expressions do not exist for the terms X_{sl} and X_{sr}. However, $(X_{sl} - X_{sr})$ can be written as a LPV representation

$$X_{sl} - X_{sr} = p(x_1)x_1 \tag{57}$$

because this term is function of roll angle x_1. Moreover, the angle of attack $\alpha(t)$ is a function of the roll angle x_1. This term can be written as a LPV representation $m(x_1)$ defined as another time-varying parameter

$$\frac{1.65}{\tan \alpha(t)} = m(x_1) \tag{58}$$

The functions $p(x_1(t))$ and $m(x_1(t))$ can be obtained by curve fitting to the experimental results (see Fig. 1 and Fig. 2).

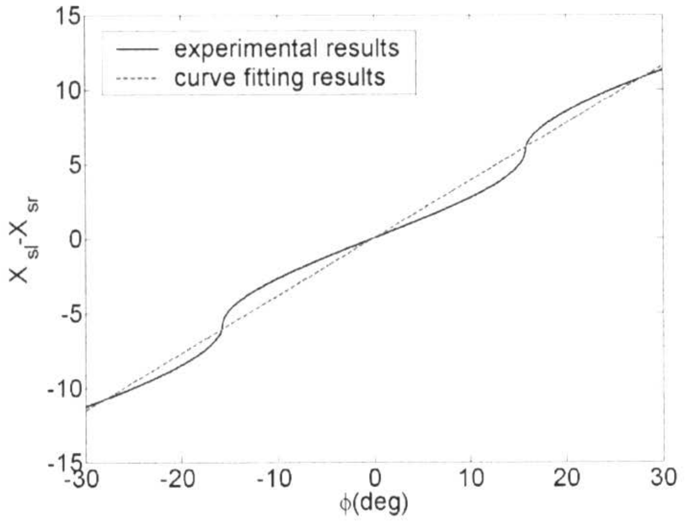


Fig. 1 Curve fitting $X_{sl}-X_{sr}$ versus the roll angle ϕ

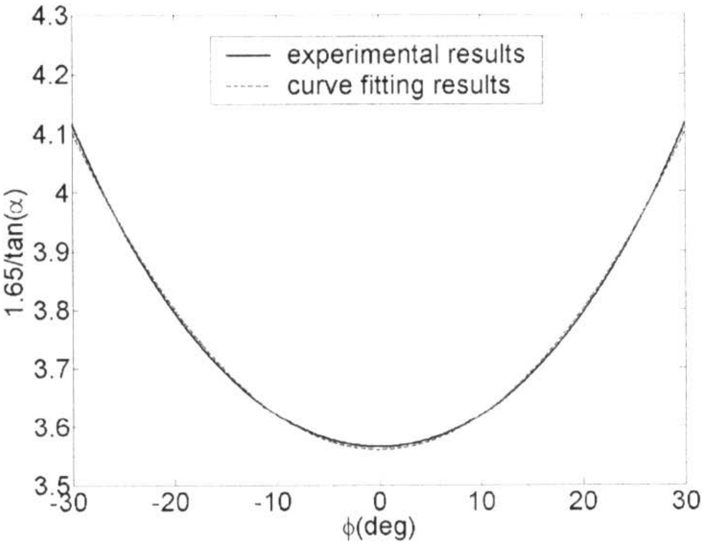


Fig. 2 Curve fitting $\frac{1.65 \cdot T}{2 \cdot \tan(\alpha)}$ versus the roll angle ϕ

$m(x_1)$ is a time-varying parameter that can be defined as

$$m(x_1) = 3.5603 + 1.9760 x_1^2 \tag{59}$$

and $p(x_1(t))$ is approximated as a constant

$$p = 22.0825 \tag{60}$$

Now, we only require considering one time-varying parameter $m(x_1)$. The parameter box of $m(x_1)$ is

$$m(x_1) \in [3.5603, \quad 4.1049] \tag{61}$$

Then, the roll moment coefficient Cl is expressed as

$$\begin{aligned} Cl &= e_0 + e_1[p(x_1)x_1 + m(x_1)(0.708x_1 + \delta)] \\ &= e_0 + e_1[p(x_1) + 0.708m(x_1)]x_1 + e_1 \cdot m(x_1)\delta \end{aligned} \tag{62}$$

The LPV representation of the dynamic for the roll motion of delta wing systems can be written as

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = &\begin{bmatrix} 0 & 1 \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ a_{h121} & 0 \end{bmatrix}\begin{bmatrix} x_1(t-\frac{T}{4}) \\ x_2(t-\frac{T}{4}) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ a_{h221} & 0 \end{bmatrix}\begin{bmatrix} x_1(t-\frac{3T}{4}) \\ x_2(t-\frac{3T}{4}) \end{bmatrix} \\ &+ \begin{bmatrix} 0 & 0 \\ a_{h321} & 0 \end{bmatrix}\begin{bmatrix} x_1(t-T) \\ x_2(t-T) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}u + \begin{bmatrix} 0 \\ c_{21} \end{bmatrix} \end{aligned} \tag{63}$$

where

$$\begin{aligned} a_{21} &= \frac{qs_a b}{I} e_1[p + 0.708m(x_1)], \\ a_{22} &= -\frac{f_c}{I}, \\ a_{h121} &= \frac{qs_a b}{I} e_1 m(x_1), \\ a_{h221} &= -\frac{qs_a b}{I} e_1 m(x_1), \\ a_{h321} &= -\frac{qs_a b}{I} 0.708 e_1 \cdot m(x_1), \\ c_{21} &= \frac{qs_a b}{I} e_0 \end{aligned} \tag{64}$$

Fig. 3 shows the comparison between experimental results of the delta wing systems and the LPV model. Fig. 4 and Fig. 5 show the states phase plot of the open-loop LPV model response to the initial condition of

$$\begin{aligned} x_1(0) &= 57.3 \text{ deg} \\ x_2(0) &= 0 \quad \text{deg/sec} \end{aligned} \tag{65}$$

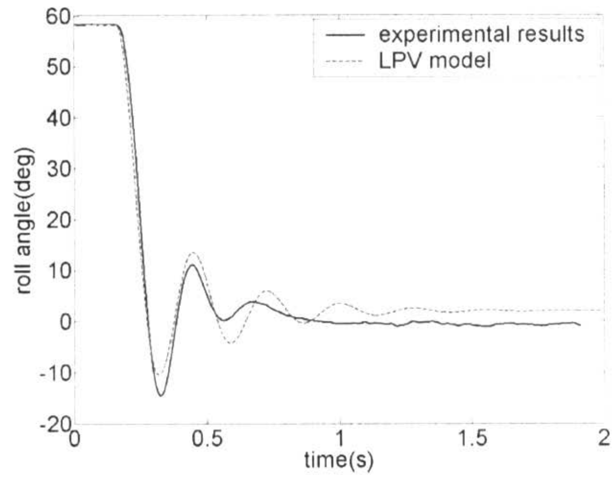


Fig. 3 Open-loop response of the LPV model vs. experimental results

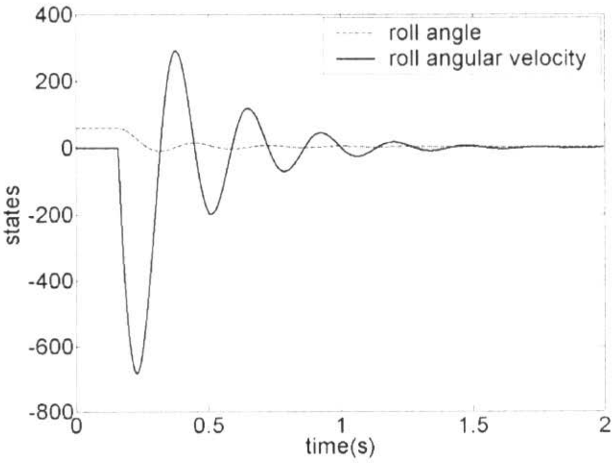


Fig. 4 States of the open-loop LPV model

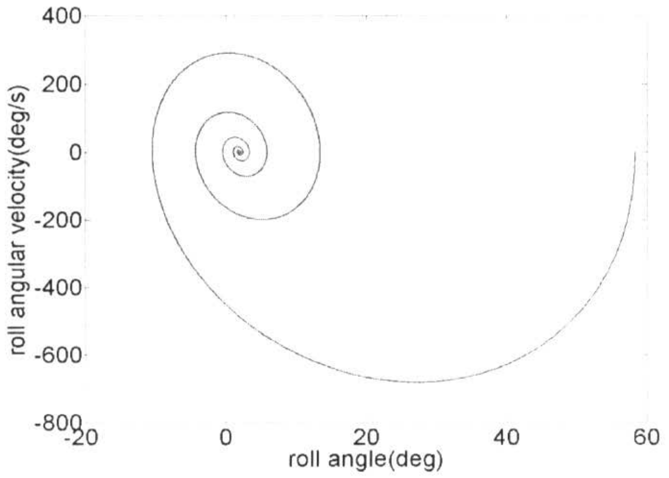


Fig. 5 Phase diagram of the open-loop LPV model

Note that the quasi-LPV model developed in this section is already in a regular form and satisfies Assumption 1 [1]. Thus we can use the LPVSMC synthesis procedure discussed in Section 3 of [1] to calculate the controller for this LPV model directly. The following section will present the LPVSMC synthesis procedure for the LPV delta wing system model.

1.4.3 Linear Parameter-Varying Sliding Mode Controller (LPVSMC) Design for Delta Wing Systems

In the previous report [1] an approach was presented for LPVSMC of vortex systems. Ref. [1] presents a linear parameter-varying sliding surface for regulation based on the regular form and synthesizes a sliding mode controller with global attractivity. The error signal is defined by

$$\mathbf{e} = \mathbf{r}(t) - \hat{\mathbf{C}}(\boldsymbol{\sigma})\mathbf{z}_1(t) \tag{66}$$

where $\mathbf{r}(t)$ is the reference input signal. The linear parameter-varying sliding surface is chosen as

$$\mathbf{s}(\boldsymbol{\sigma}) = \mathbf{S}(\boldsymbol{\sigma}, \mathbf{z}_1) + \mathbf{z}_2 \tag{67}$$

where $\mathbf{S}(\boldsymbol{\sigma}, \mathbf{z}_1)$ is a linear parameter-varying operator of $\mathbf{z}_1$. The LPV operator $\mathbf{S}(\boldsymbol{\sigma}, \mathbf{z}_1)$ is similar to the function $\mathbf{S}(\mathbf{z}_1)$, but $\mathbf{S}(\boldsymbol{\sigma}, \mathbf{z}_1)$ adds time-varying parameters of the plant. The parameter-varying controller can adapt itself to the variation of the system parameters on-line. Therefore the LPVSMC approach can potentially obtain better control performance and less conservative results than standard SMC with a constant parameter sliding surface. The chosen switching function $\mathbf{s}$ will have some dynamics compared to a conventional switching function in terms of the defined new variable $\mathbf{z}_w$

$$\dot{\mathbf{z}}_w = \mathbf{F}(\boldsymbol{\sigma})\mathbf{z}_w + \mathbf{G}(\boldsymbol{\sigma})\mathbf{r} - \mathbf{G}(\boldsymbol{\sigma})\hat{\mathbf{C}}(\boldsymbol{\sigma})\mathbf{z}_1 \tag{68}$$

$$\mathbf{S}(\boldsymbol{\sigma}, \mathbf{z}_1) = \mathbf{H}(\boldsymbol{\sigma})\mathbf{z}_w + \mathbf{L}(\boldsymbol{\sigma})\mathbf{r} - \mathbf{L}(\boldsymbol{\sigma})\hat{\mathbf{C}}(\boldsymbol{\sigma})\mathbf{z}_1$$

The LPVSMC control law is given by

$$\mathbf{u}(t) = \begin{cases} \mathbf{u}_{eq} - \mathbf{u}_n & \mathbf{s}^T\hat{\mathbf{B}} > 0 \\ \mathbf{u}_{eq} + \mathbf{u}_n & \mathbf{s}^T\hat{\mathbf{B}} < 0 \end{cases} \tag{69}$$

where

$$\mathbf{u}_n = \frac{k\|\mathbf{s}\|}{\|\mathbf{s}^T\hat{\mathbf{B}}\| + \varepsilon}\left(\mathbf{s}^T\hat{\mathbf{B}}\right)^T \tag{70}$$

$$\begin{aligned} \mathbf{u}_{eq} = -\hat{\mathbf{B}}^{-1}\Big\{ & \left[\frac{\partial \mathbf{H}(\boldsymbol{\sigma})}{\partial \boldsymbol{\sigma}}\dot{\boldsymbol{\sigma}} + \mathbf{H}(\boldsymbol{\sigma})\mathbf{F}(\boldsymbol{\sigma}) + \mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma})\hat{\mathbf{A}}_{12}(\boldsymbol{\sigma})\mathbf{H}(\boldsymbol{\sigma}) - \hat{\mathbf{A}}_{22}(\boldsymbol{\sigma})\mathbf{H}(\boldsymbol{\sigma})\right]\mathbf{z}_w \\ & + \Big[-\mathbf{H}(\boldsymbol{\sigma})\mathbf{G}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma}) - \frac{\partial \mathbf{L}(\boldsymbol{\sigma})}{\partial \boldsymbol{\sigma}}\dot{\boldsymbol{\sigma}}\mathbf{C}(\boldsymbol{\sigma}) - \mathbf{L}(\boldsymbol{\sigma})\frac{\partial \mathbf{C}(\boldsymbol{\sigma})}{\partial \boldsymbol{\sigma}}\dot{\boldsymbol{\sigma}} - \mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma})\hat{\mathbf{A}}_{11}(\boldsymbol{\sigma}) \\ & \quad - \mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma})\hat{\mathbf{A}}_{12}(\boldsymbol{\sigma})\mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma}) + \hat{\mathbf{A}}_{21}(\boldsymbol{\sigma}) + \hat{\mathbf{A}}_{22}(\boldsymbol{\sigma})\mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma})\Big]\mathbf{z}_1 \\ & + \left[\frac{\partial \mathbf{L}(\boldsymbol{\sigma})}{\partial \boldsymbol{\sigma}}\dot{\boldsymbol{\sigma}} + \mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma})\hat{\mathbf{A}}_{12}(\boldsymbol{\sigma})\mathbf{L}(\boldsymbol{\sigma}) - \hat{\mathbf{A}}_{22}(\boldsymbol{\sigma})\mathbf{L}(\boldsymbol{\sigma}) + \mathbf{H}(\boldsymbol{\sigma})\mathbf{G}(\boldsymbol{\sigma})\right]\mathbf{r} \\ & + \left[-\mathbf{L}(\boldsymbol{\sigma})\mathbf{C}(\boldsymbol{\sigma})\hat{\mathbf{A}}_{h11}(\boldsymbol{\sigma}) + \hat{\mathbf{A}}_{h21}(\boldsymbol{\sigma})\right]\mathbf{z}_1(t-h) \\ & + \hat{\mathbf{A}}_{h22}(\boldsymbol{\sigma})\mathbf{z}_2(t-h) \\ & + \mathbf{L}(\boldsymbol{\sigma})\dot{\mathbf{r}}(t)\Big\} \end{aligned} \tag{71}$$

In the following section the approach is extended for SMA actuators.

1.4.4 Simulation Results for the Vortex-Coupled Delta Wing System with the Shape Memory Alloy (SMA) Micro-Actuator [2]

To include the SMA dynamics in the delta wing roll dynamics, SMA model is expressed as

$$\begin{aligned} \dot{x} &= Ax + Bu_1(t-\tau_a) \\ \dot{u}_1 &= -\frac{1}{\varepsilon_1}u_1 + \frac{1}{\varepsilon_1}u_2 \\ \dot{u}_2 &= -\frac{1}{\varepsilon_2}u_2 + \frac{1}{\varepsilon_2}u \end{aligned} \tag{72}$$

The states are defined as

$$x_1 = \Phi(t), x_2 = \dot{\Phi}(t), x_3 = u_1 \quad \text{and} \quad x_4 = u_2 \tag{73}$$

The LPV representation of the dynamic for the roll motion of delta wing systems can be written as

$$\begin{aligned} \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & -\frac{1}{\varepsilon_1} & \frac{1}{\varepsilon_1} \\ 0 & 0 & 0 & -\frac{1}{\varepsilon_2} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ a_{h121} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t-\frac{T}{4}) \\ x_2(t-\frac{T}{4}) \\ x_3(t-\frac{T}{4}) \\ x_4(t-\frac{T}{4}) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ a_{h221} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t-\frac{3T}{4}) \\ x_2(t-\frac{3T}{4}) \\ x_3(t-\frac{3T}{4}) \\ x_4(t-\frac{3T}{4}) \end{bmatrix} \\ &+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ a_{h321} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t-T) \\ x_2(t-T) \\ x_3(t-T) \\ x_4(t-T) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t-\tau_a) \\ x_2(t-\tau_a) \\ x_3(t-\tau_a) \\ x_4(t-\tau_a) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{\varepsilon_2} \end{bmatrix} u \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} \end{aligned} \tag{74}$$

where

$$a_{21}=\frac{qs_ab}{I}e_1[p+0.708m(x_1)], a_{22}=-\frac{f_c}{I},$$
$$a_{h121}=\frac{qs_ab}{I}e_1m(x_1), a_{h221}=-\frac{qs_ab}{I}e_1m(x_1), a_{h321}=-\frac{qs_ab}{I}0.708e_1\cdot m(x_1), \quad (75)$$
$$\varepsilon_1=\varepsilon_2=0.02, \ \tau_a=0.01$$

Running 'PDLKLMIsForDeltaWing4zw3.m', we can obtain the following results:

```
eigP0 =   2.6068    2.6068    2.6068    2.6068    2.6068    2.6068
eigP1 =   3.1514    3.1514    3.1514    3.1514    3.1514    3.1514

eigQ0 =    34620     35508     39590     48157  1.65e+005 9.6502e+005
eigQ1 =    34620     35508     39590     48157  1.65e+005 9.6502e+005

eigR0 =   28.231    115.51    390.55    547.54    3280.2     41055
eigR1 =   28.231    115.51    390.55    547.54    3280.2     41055

LMI_F =  -4    0    0  -18    0    0
          0   -4    0  -18    0    0
          0    0   -4  -18    0    0
         10  -10   10   10    0    0
```

By using the 'LyapunovDeltawing.m', the simulation results can be obtained. In the program, we call 'f_LyapunovKrasoskii_dot.m' to calculate the v dot. This function using the alternative extended Simpson's rule to approximate the integral. Figures 6 to 12 show the simulation result for the delta wing model coupled with 2D SMA model. Figures are showing acceptable results for the controller applied to the system with SMA inputs.

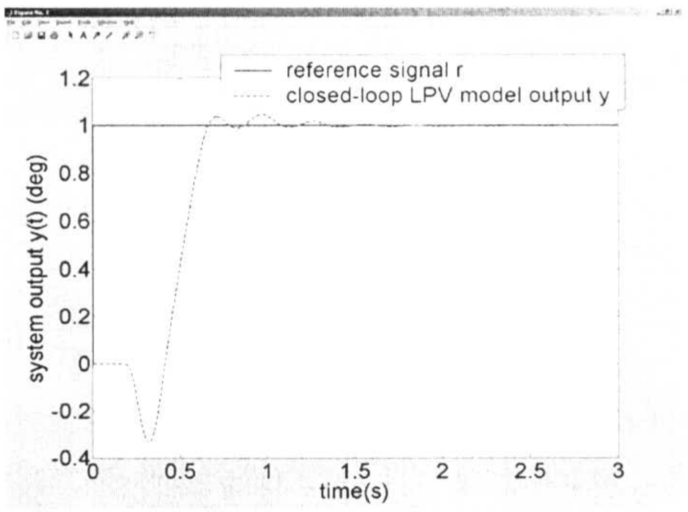


Fig. 6 output time history

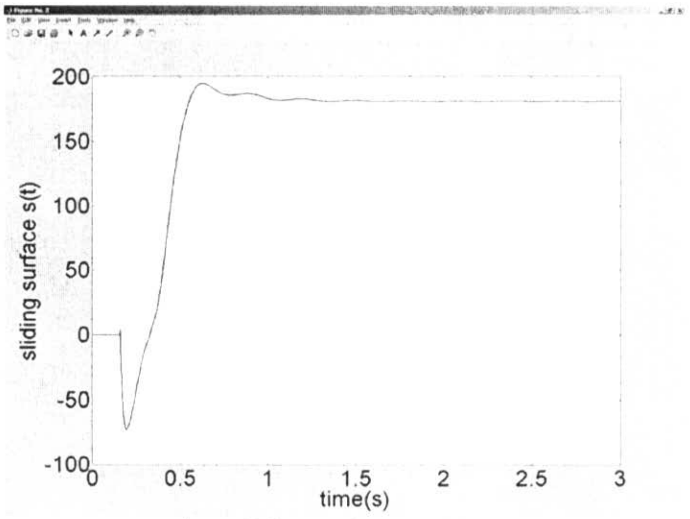


Fig. 7 sliding surface time history

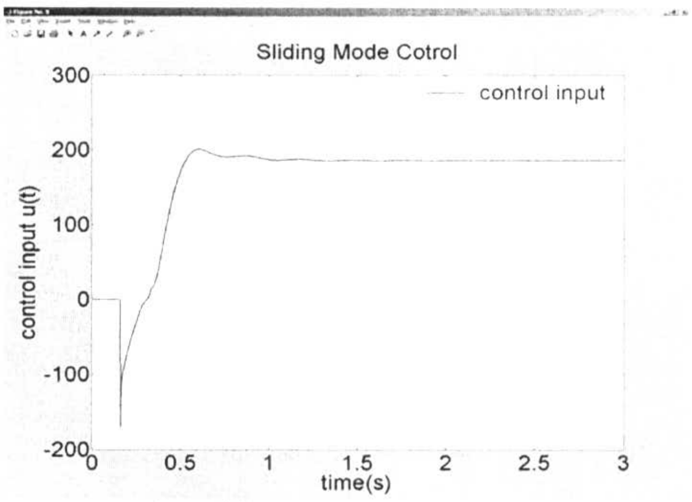


Fig. 8 sliding control input time history

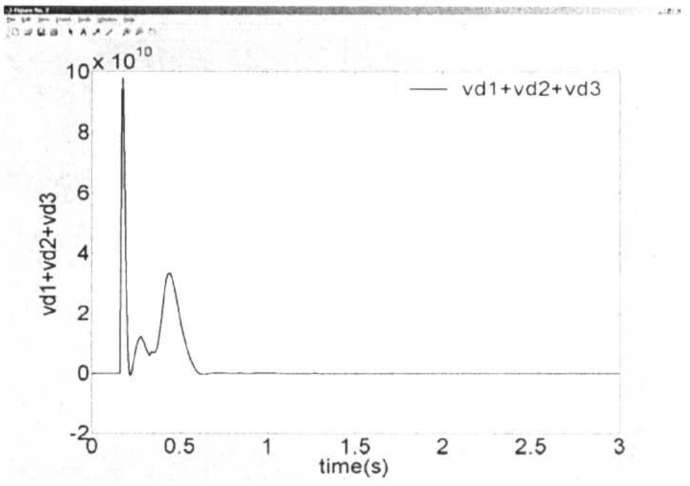


Fig. 9 (vd1+vd2+vd3) time history

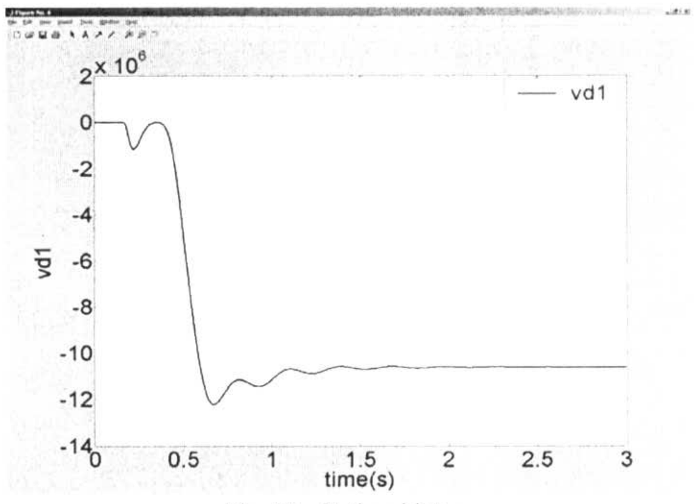


Fig. 10 vd1 time history

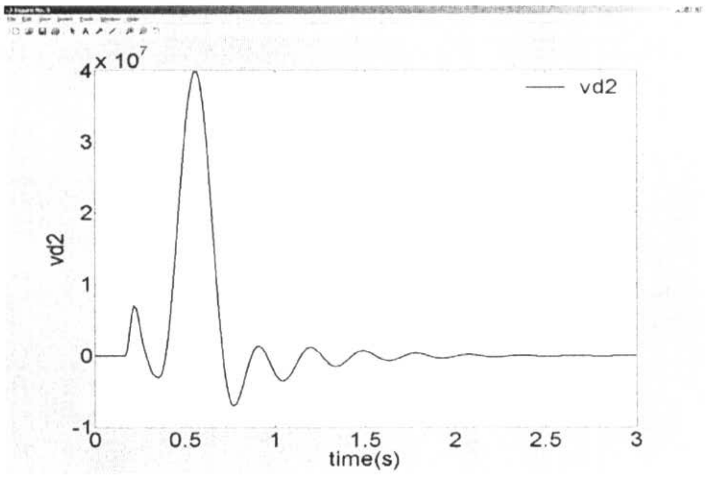


Fig. 11 vd2 time history

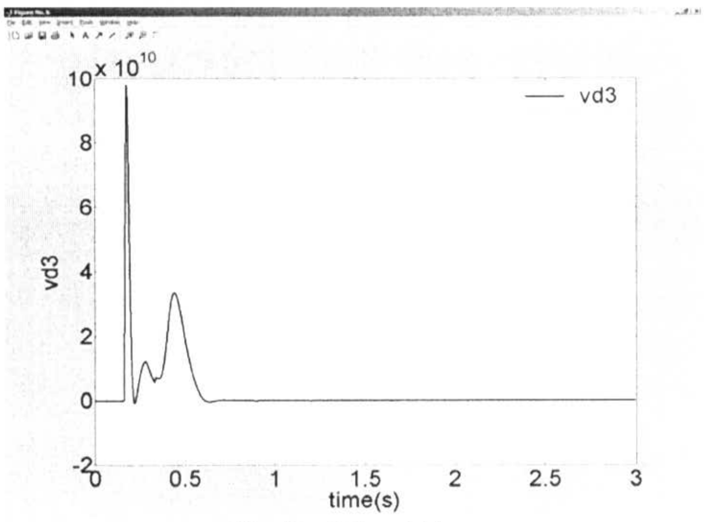


Fig. 12 vd3 time history

1.4.5 References

[1] Yang, M., Gordon, B. W., "Linear Parameter-Varying Sliding Mode Control of State Delayed Systems with Application to Delta Wing Vortex Coupled Dynamics", Internal Report (No. 4), submitted to Dr. C. A. Rabbath, Defence Research and Development Canada (DRDC), Valcartier, Quebec, Canada, Nov. 2005.

[2] Yang, M., Gordon, B. W., "Simulation Results for the Vortex-Coupled Delta Wing System with the Shape Memory Alloy (SMA) Micro-Actuator", Internal Report, submitted to Dr. C. A. Rabbath, Defence Research and Development Canada (DRDC), Valcartier, Quebec, Canada, Dec. 2005.

1.5 <u>Section 5:</u> Conclusions and Future Work

Conclusions

The multiple neural network adaptive control approach based on a hybrid system framework has been developed and applied to the vortex-coupled delta wing dynamics in the presence of uncertainties, probable unmodelled dynamics and state delay. We defined three different RBF adaptive controllers for three different regions. The regions are defined based on the roll angle of delta wing. Then we tuned each controller for each region for the best performance. To handle the chattering we used hysteresis switching logic, using this technique we can minimize the chattering in our closed loop system. The simulation results show that the proposed switching controller can be used for tracking control of delta wing on sophisticated trajectories.

The combinatory robust adaptive control scheme has been applied to the delta wing dynamics coupled with the new SMA micro actuator dynamics. Two different control loops has been implemented for SMAs on the left and right sides of the wing. The outputs of these control loops are the perturbations added to the vortex breakdown positions. The simulations show acceptable results with the new dynamics. However we need to tune the controller for more perturbations on the wing.

The LPVSMC approach is applied to a LPV representation of vortex-coupled delta wing system dynamics. It is also demonstrated that the LPVSMC approach is less conservation than fixed parameter SMC and Gouaisbaut's SMC approach. Furthermore, it is shown from the simulation results that the LPVSMC performance for the nonlinear system is comparable or even better than for the LPV model. This indicates the approach is also effective for the original nonlinear system dynamics. The approach presented is one of the first applications of nonlinear control for vortex-coupled delta wing systems that is verified on the system with SMA actuator dynamics.

Future Work

For hybrid neural network adaptive control we can define new parameters for our switching logic. Switching event should be defined better for better performance of the controller. It is also possible to use different neural networks like wavelets in the controller structure. The comprehensive stability proof of the switching control should be carried out.

For the control of the delta wing coupled with the new SMA micro actuator dynamics we need to come up to the best control structure for the perturbations on the wing to get the best control results for the delta wing complicated roll manoeuvres. The controller needs more tuning.

Since the LPVSMC controller coefficients depend directly on parameters to be measured or estimated online, it is anticipated that filtering methods to deal with parameter measurement noise or rapid parameter variations might be necessary in the future. Another future work is to provide a more rigorous analysis of the robustness properties of the LPVSMC approach, since one of the advantages of SMC is to achieve robustness and disturbance rejection. The robustness investigation can be divided into two steps: the robustness of the attraction of sliding surface and the robustness on the sliding surface. Finally, it is anticipated that this method will lead to significant improvement over existing SMC approaches in aerospace and automotive applications with parameter variations. The new method should be applied to such applications to get more experience and help guide the improvement of the method in the future.

CHAPTER 2: INVESTIGATION OF TIME DELAY SHAPE MEMORY ALLOY (SMA) ACTUATORS USING RECEDING HORIZON CONTROL APPROACH FOR VORTEX BREAK DOWN AUGMENTED ROLL DYNAMICS OF DELTA WING AIRCRAFT

(Task # 3)

Hojjat A. Izadi
PhD Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory
Department of Mechanical and Industrial Engineering
Concordia University
Montréal, Québec, Canada H3G 1M8

March 2006

Abstract

This report documents the work related to task #3 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". Some advantages of Shape Memory Alloy (SMA) actuator make it an attracting choice for flow control problems; however, it involves some nonlinearity like time delay. In this report, Receding Horizon Control (RHC) method is used to investigate the effects of the SMA's time delay on the behaviour of system. A nonlinear dynamics of the delta wing aircraft which augments the "vortex coupled break down location" for the roll manoeuvre is also used; in this nonlinear dynamics, vortex location on the wings is controlled by SMA actuators. Simulation results show that the RHC can control the vortex coupled dynamics of delta wing for small enough time delay ($\tau \leq 0.25$).

Nomenclature

b	wing span	K	controller gain
P,Q,R	Weight matrices of quadratic cost function	α	angle of attack
q	dynamic pressure	T	finite horizon time
I	moment of Inertia	ϕ	roll angle
X_{vb}	Non-dimensional vortex breakdown location	fc	friction coefficient
Xs	static term in X_{vb}	s	wing element area
Cl	roll moment coefficient	μ	Uncertainty coefficient
d	State delay	τ	SMA actuator delay

Subscripts

w	wing	VB	vortex breakdown
l	left wing vortex	r	right wing vortex

Abbreviations

RHC:	Receding Horizon Control
SMA:	Shape Memory Alloy
MIMO:	Multi-Input Multi-Output

2.1 Introduction

This report addresses the following task related to the project "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques":

Task 3: Integrate outer-loop flow control schemes with inner-loop shape memory alloy actuators.

Shape Memory Alloy (SMA) actuators have some prominent abilities that make them an attracting choice for control of flow control problems; more notable, they offer a more compact, solid and cheaper actuators. However, SMAs have some serious nonlinearity such as backlash-like hysteresis. The main idea to study this backlash is to consider it as a delay in the output of SMA; then, this delay will be studied as a bifurcation parameter while we will observe how its quantitative changes will affect the qualitative behaviour of system. Receding horizon controller (RHC) due to its unique capability to handle the constraints and nonlinearity is a good candidate to control the delay dependent SMA actuators. The MIMO dynamics of vortex-coupled roll dynamics of delta wing aircraft is used to study the effect of the time delay in the control loop and integrating the SMA with nonlinear dynamics of vortex break down delta wing aircraft.

In this report, first the nonlinear dynamics of "vortex-coupled roll dynamics of delta win aircraft" is introduced. After that the theoretical issues of a robust quasi-RHC is presented and finally a quasi-RHC controller is designed and implemented to the "vortex-coupled roll dynamics of delta wing aircraft".

2.2 MIMO Dynamics of Vortex coupled Delta Wing with SMA

The "roll dynamics of delta wing aircraft augmenting a vortex coupled break-down location" is a nonlinear dynamics containing state delay, uncertainty, friction and zero dynamics with a weak stability behaviour of the open loop system. The roll dynamics of delta wing aircraft augmenting vortex coupled dynamics is described by [1]:

$$\begin{cases} \dot{x}_1(t) = c\,x_2(t) \\ \dot{x}_2(t) = -c\,x_1(t) - \varepsilon_d\,x_2(t) + x_4(t) + x_4(t-d) \\ \dot{x}_3(t) = -\varepsilon_d\,x_3(t) + x_4(t) \\ \dot{x}_4(t) = u(t)/I_w - Cl(Xvb(x(t)))Q - f_c\,x_4(t) \\ \dot{x}_5(t) = (-x_5(t) + h_2\rho_{vbl}u_2(t-\tau))/\varepsilon_l \\ \dot{x}_6(t) = (-x_6(t) + h_3\rho_{vbr}u_3(t-\tau))/\varepsilon_r \end{cases} \tag{1}$$

To incorporate the dynamics of shape memory alloy (SMA) actuators the 5th and 6th equations are added to the system dynamics [1], hence the main focus here is the two last equations; also, τ is the amount of delay induced to the system and will be discussed completely in this report.

The rolling moment coefficient C_l is a nonlinear function of vortex breakdown locations [2]; we assume it is a 3rd order polynomial in the following form:

$$C_l(X_{vbl},X_{vbr}) = e_0 + e_1(X_{vbl}-X_{vbr}) + e_2(X_{vbl}^2 - X_{vbr}^2) + e_3(X_{vbl}^3 - X_{vbr}^3) \tag{2}$$

Where, X_{vbl} and X_{vbr} represent the vortex breakdown locations for the left and right vortices, and e_0, e_1, e_2 and e_3 are constant coefficients obtained using 3rd order least square curve fitting of the experimental data. Vortex breakdown locations are considered as following [2]:

$$X_{vbl}(t) = X_{sl}(t) + X_{sl}\,k_q(t)\,x_2(t) \tag{3}$$

$$X_{vbr}(t) = X_{sr}(t) + X_{sr}\,k_q(t)\,x_2(t) \tag{4}$$

Where, X_{sl} and X_{sr} are static components of vortex breakdown locations in the overall vortex breakdown locations [3], and:

$$k_q(t) = 0.91 / \tan(\alpha(t)) \tag{5}$$

Where, angle of attack (α) is computed as follows:

$$\alpha(t) = \operatorname{atan}(\cos(\ x_1(t)) \cdot \tan(\ \sigma\ )) \tag{6}$$

And, bevel angle of the wing (σ) is an experimentally obtained value [3]. Equation (1) describes the vortex break down roll dynamics of delta wing aircraft. For a delta wing aircraft, the following bounds are considered on the bank angle, roll rate and control input:

$$\begin{cases} -100^\circ \le \phi \le 100^\circ \\ -100 \text{ deg/sec} \le \dot{\phi} \le 100 \quad \text{deg/sec} \\ -10 \le u \le 10 \end{cases} \tag{7}$$

Then the following bounds can be considered on the states and control inputs, these are selected from the simulation results of [2]:

$$\begin{cases} -0.4 \le x_1 \le 0.4 \\ -0.4 \le x_2 \le 0.4 \\ -1.74 \le x_3 \le 1.74 \ \ (Rad.) \\ -1.74 \le x_4 \le 1.74 \ \ (Rad.) \\ -1 \le x_5 \le 1 \\ -1 \le x_6 \le 1 \\ -10 \le u_1 \le 10 \\ -1 \le u_2 \le 1 \\ -1 \le u_3 \le 1 \end{cases} \tag{8}$$

Choosing above parameters according to [2], we have:

$$\begin{cases} c = 31.4159 \\ \varepsilon_d = 0.1 \\ I_w = 0.0305 \\ f_c = 0.4 \\ h_1 = 1 \\ \overline{Q} = 27658.97 \\ h_2 = h_3 = 0.5 \\ \rho_{vbl} = \rho_{vbr} = 0.08 \\ \varepsilon_{SMA} = 0.1 \\ d = 0.1 \end{cases} \tag{9}$$

2.3 Quasi Receding Horizon Control

The RHC method is based on a repeatedly on-line solution to a "finite horizon open loop optimal control" problem. An optimal control problem is solved for a period of time called the *prediction horizon*. Then, the first portion or the second portion of the computed optimal input may be applied to the plant in a period of time called the *execution horizon* until the next sampling of the states becomes available [4].

Some benefits of the RHC make it a good candidate to be used for the flow control problem of aerospace vehicles which are nonlinear (like delay), constrained and uncertain dynamics. For instance, by the means of RHC, it is possible to design a feedback control law for nonlinear systems. Excellent capability to handle the constraints is another unique advantage of the RHC. The optimality of the RHC is also a considerable benefit.

Suppose that the nominal model is presented as the following general form:

$$\dot{x} = f\left(x(t), u(t)\right) \qquad x(0) = x_0 \tag{10}$$

Where $\mathbf{x}(t) \in \Re^n$ is the state of the system and $\boldsymbol{u}(t) \in \Re^m$ is the input vector satisfying the constraints:

$$u(t) \in U \qquad \forall t \geq 0 \tag{11}$$

U is the set of allowable inputs. In order to meet the necessary conditions for stability; it is also assumed that:

A1- f is twice continuously differentiable.

A2- U is compact and convex and contains origin.

A3- system (10) has a unique solution for a given initial condition [4].

Then, under these circumstances the RHC is the repeated solution of the following problem [5]:

Problem 1: Find

$$\min_{\mathbf{u}(.)} \int_0^T q(x(\tau), u(\tau))d\tau + V(x(T)) \tag{12}$$

Subject to

$$\dot{x}(t) = f\left(x(t), u(t)\right) \qquad x(0) = x_0 \tag{13}$$

$$lb_0 \leq \psi(x(0), u(0)) \leq ub_0 \tag{14}$$

$$lb_t \le S(x(t),u(t)) \le ub_t \tag{15}$$

$$x(T) \in \Omega_r \tag{16}$$

Where

$$\Omega_r = \{x \in \Re^n : V(x) \le r\} \tag{17}$$

ψ is an initial constraint on the parameters and S is a trajectory constraint enforced over the entire time interval. lb and ub are the lower and upper bounds respectively on the constraints.

Constraint (16) is added to guarantee the stability of the RHC; in fact, by the means of the RHC the trajectory of system is driven to a neighbourhood of the origin and after that a linear local feedback controller stabilizes the system; hence, for the open loop system, control input is as follows:

$$u(s) = \begin{cases} u^*(s; x(t), t) & x \in X \cap \Omega_r^c \\ Kx(s) & x \in \Omega_r \end{cases} \tag{18}$$

Where, X is the set of initial states that there exists a feasible solution for them, Ω_r^c is the set of states which do not belong to Ω_r and K is the gain of local feedback controller.

Repeating this computation yields a closed-loop feedback control law. For a receding horizon sampling period δ when δ belongs to (0, T]; then, the closed-loop system is represented as:

$$\begin{aligned} &\dot{x}(\tau) = f\left(x(\tau), u^*(\tau)\right) \\ &u^*(\tau) = u^*(\tau; x(t)) \qquad \tau \in [t, t+\delta), \quad 0 < \delta \le T \end{aligned} \tag{19}$$

Where $u^*(\tau; x(t)), \quad \tau \in [t, t+T]$, is the optimal control obtained from the above problem with the initial condition ***x(t)***, and t is the start time of the optimization process and the instant at which states are sampled.

One suitable choice for cost function (12) is the quadratic cost functions, because of their unique properties in the optimization problems [6]. With the following selections, the RHC problem becomes quadratic [5]:

$$q(x(t),u(t)) = \frac{1}{2}x^T(t)Qx(t) + \frac{1}{2}u^T(t)Ru(t) \tag{20}$$

And:

$$V(x(T)) = \frac{1}{2}x^T Px \tag{21}$$

$Q,P \in \Re^{n\times n}$ and $R \in \Re^{m\times m}$ denote positive-definite, symmetric weighting matrices, T is a finite prediction time and $\boldsymbol{x}(t;\boldsymbol{x_0})$ denotes the trajectory of the system (10) driven by $u(t)$ starting from the initial condition x_0. Since states and control inputs implicitly depend on time, it is required to parameterize the states and control inputs of system as a function of time to solve this optimization problem. The polynomial basis function has been used for parameterization of states and control inputs.

2.3.1 RHC Design for MIMO Roll Dynamics

One of the most efficient tools for decreasing the computation effort in optimization problems is the flat output. The main idea is to reduce the dimension of the optimal control problem to a lower dimension that easing the formulation and reducing the computational burden for optimization process. The motivations for using flat outputs in optimization problems are discussed completely in the previous reports and interested readers are referred to [6] for more detail information and application of flat outputs in the trajectory generation and optimization problems.

To design RHC for vortex coupled delta wing dynamics of section 1, consider the following flat outputs:

$$\begin{cases} z_1 = x_1 \\ z_2 = x_3 \\ z_3 = x_5 \\ z_4 = x_6 \end{cases} \tag{22}$$

Then, all states and inputs of the system can be recovered as a function of above flat outputs and their derivatives:

$$\begin{cases} x_1(t) = z_1(t) \\ x_2(t) = \dot{z}_1(t)/c \\ x_3(t) = z_2(t) \\ x_4(t) = \dot{z}_2(t) + \varepsilon_d z_2(t) \\ x_5(t) = z_3(t) \\ x_6(t) = z_4(t) \\ \\ u_1(t) = I_w[(\ddot{z}_2(t) + \varepsilon_d \dot{z}_2(t)) + f_c(\dot{z}_2(t) + \varepsilon_d z_2(t)) + C_l.\overline{Q}] \\ u_2(t) = \dfrac{\varepsilon_{SMA}}{h_2 \rho_L}(\dot{z}_3(t) + z_3(t)) \\ u_3(t) = -\dfrac{\varepsilon_{SMA}}{h_3 \rho_r}(\dot{z}_4(t) + z_4(t)) \end{cases} \tag{23}$$

The matrix penalties are selected as following:

$$Q = I$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

$$P = \begin{bmatrix} 8.0787 & 0.0540 & 0.5265 & -0.2271 & 0.0000 & -0.0000 \\ 0.0540 & 8.5553 & 0.4820 & 0.2856 & 0.0000 & -0.0000 \\ 0.5265 & 0.4820 & 9.6056 & 0.2916 & 0.0000 & -0.0000 \\ -0.2271 & 0.2856 & 0.2916 & 0.0559 & 0.0000 & -0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 2.9802 & -0.0000 \\ -0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 & 2.9802 \end{bmatrix} \tag{24}$$

Where, I is the identity matrix. P is obtained from the Lyapanov equation for the linerized of delta wing dynamics, and also, Q and R are tuned based on a trial and error procedure for the best response of system.

It is obvious that with larger finite horizon time T, the solution of RHC will get closer to the optimal solution; on the other hand, high T will increase the computational burden; hence, selection of T is a trade off between computation effort and optimality. For our problem by a trail and error procedure the optimal finite horizon time is chosen as $T=1$ *Sec*.

Also, the execution time should be small as much as possible to meet the necessary conditions for stability and performance; but, in reality, it is limited by sampling time and the time required for optimization. Actually, it has to be bigger than sampling time and optimization time. In our case, the execution time is chosen as sec; in fact, it is assumed that the sampling time and the required time for optimization are smaller than 0.1 sec.

The quasi robust RHC is applied to the actual vortex coupled delta wing model with all its nonlinearities including delay and uncertainties in the lift coefficient; to study the robustness of the controller to the disturbances, a noise with magnitude of 10% of the amount of states is added to the actual model. Piecewise polynomial class functions have been used to estimate the flat outputs as a function of time; the order of polynomials is three with second-order continuity at break points.

For implementation of the delay term in both states and control inputs, a buffer is used to restore the amount of the delayed states and control inputs.

2.3.2 Simulation Results for Different Time Delay τ

The simulation is done by Matlab software and the related code is "*RHC_Vortex_SMAdelay.m*" that invokes two functions called: "*jfun_closeloop.m*" and *"model_f_nlinear_delay.m"*. The first function represents the cost function to be minimized by RHC and the second function calculates the lift coefficient from equation (2); this function needs a data file called *"COEF_nonlin.mat"* containing the coefficients of the polynomial which is an approximation to lift coefficient (2). Complementary comments within the m-files help user to understand each part of the code.

Simulation results for different time delays are depicted in figures 1 through 24; the time history of all states, control inputs and cost function are shown for an initial condition. It can be seen from these simulation results that as τ increases, the 5th and 6th state trajectories start to oscillate (figures 5 to 12). Around $\tau = 0.3$ (figures 13 to 16), it has a periodic behaviour like a limit cycle and as τ goes up, it will get unstable gradually (figures 17 to 20).

Although a bigger τ makes the 5th and 6th states unstable (figures 21 to 24), it has no effect on the other states. However, this can affect the cost function; in that, for higher τ, the cost function may not decrease strictly (figures19 and 23).

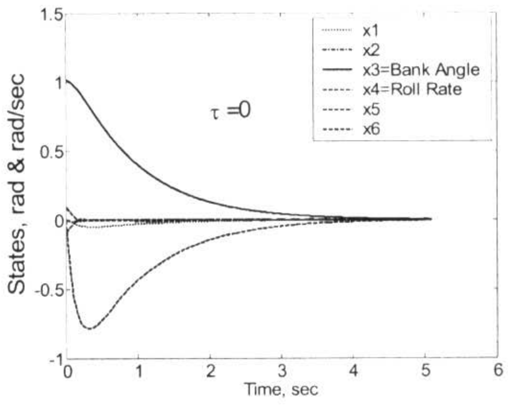


Fig.1. Time history of states for closed-loop MIMO system for $\tau = 0$

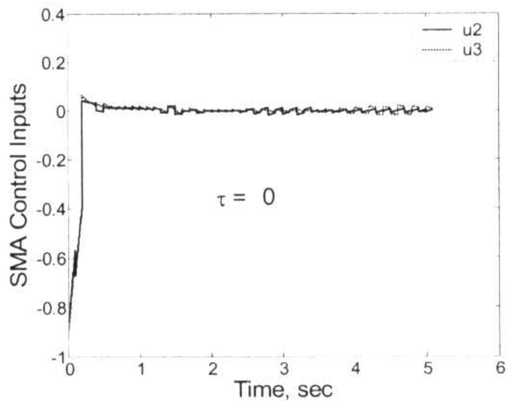


Fig.2. Time history of inputs to SMA for $\tau = 0$

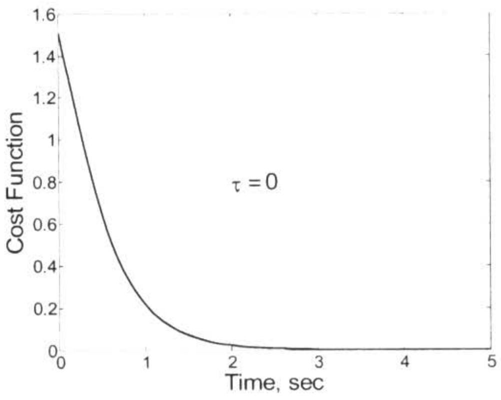


Fig.3. Time history of cost function for $\tau = 0$

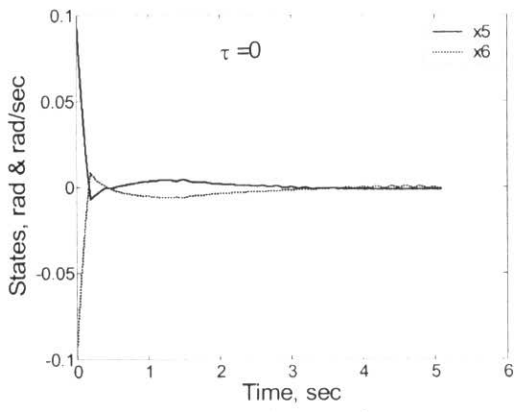


Fig.4. Time history of 5th and 6th states for $\tau = 0$

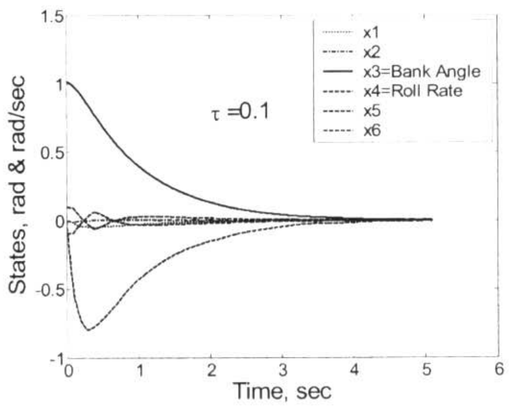


Fig.5. Time history of states for closed-loop MIMO system for $\tau = 0.1$

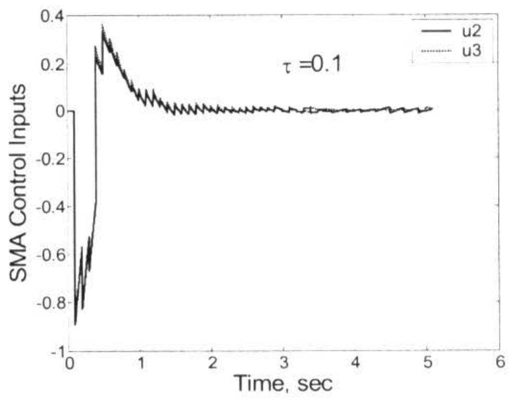


Fig.6. Time history of inputs to SMA for $\tau = 0.1$

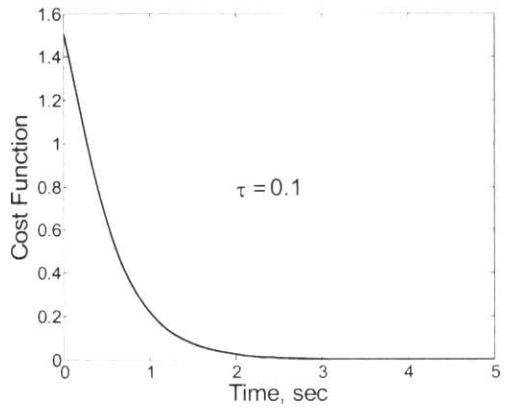


Fig.7. Time history of cost function for $\tau = 0.1$

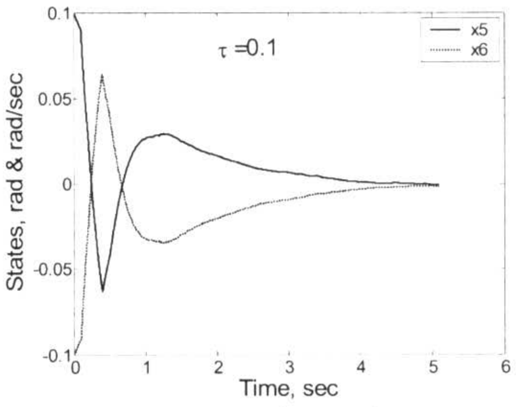


Fig.8. Time history of 5th and 6th states for $\tau = 0.1$

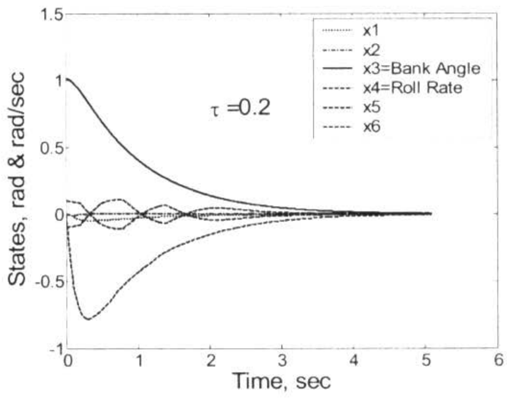


Fig.9. Time history of states for closed-loop MIMO system for $\tau = 0.2$

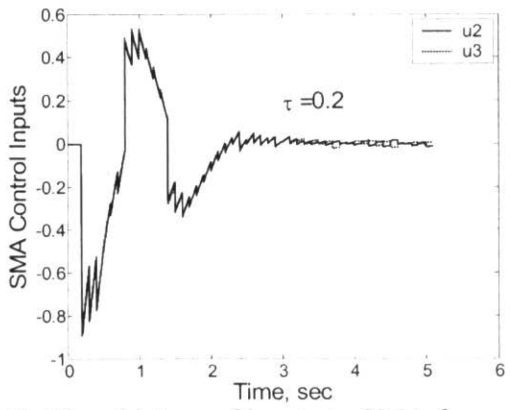


Fig.10. Time history of inputs to SMA for $\tau = 0.2$

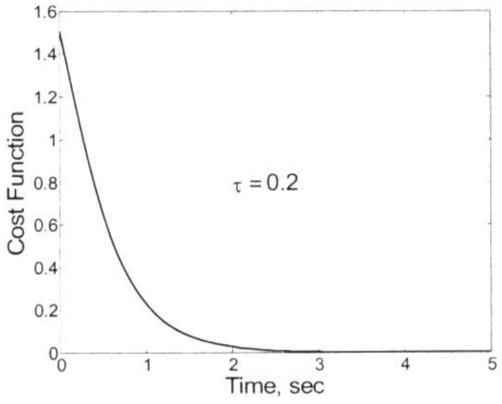


Fig.11. Time history of cost function for $\tau = 0.2$

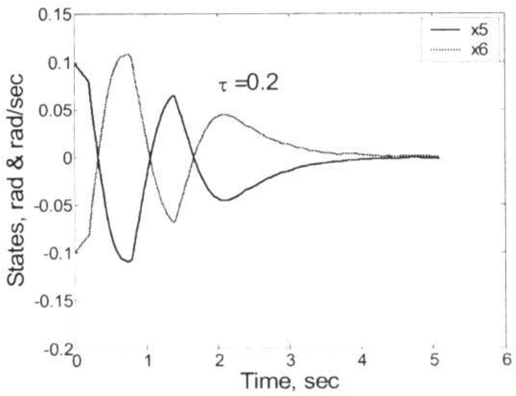


Fig.12. Time history of 5th and 6th states for $\tau = 0.2$

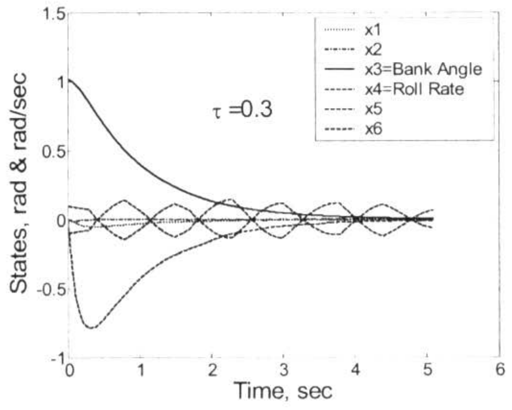


Fig.13. Time history of states for closed-loop MIMO system for $\tau = 0.3$

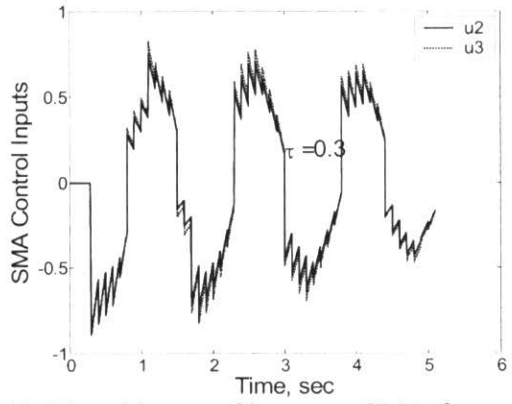


Fig.14. Time history of inputs to SMA for $\tau = 0.3$

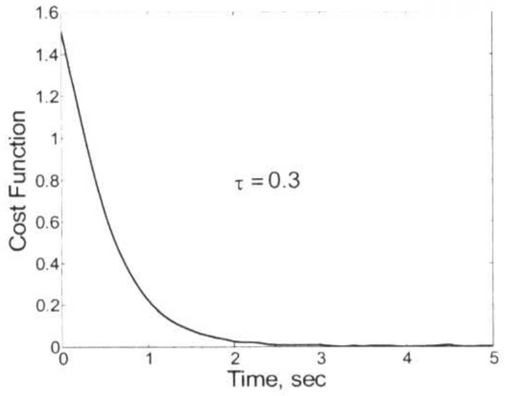


Fig.15. Time history of cost function for $\tau = 0.3$

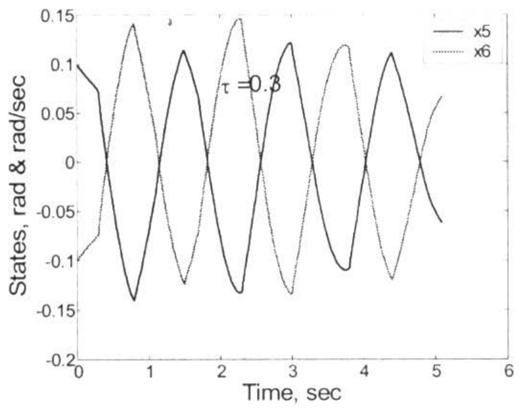


Fig.16. Time history of 5th and 6th states for $\tau = 0.3$

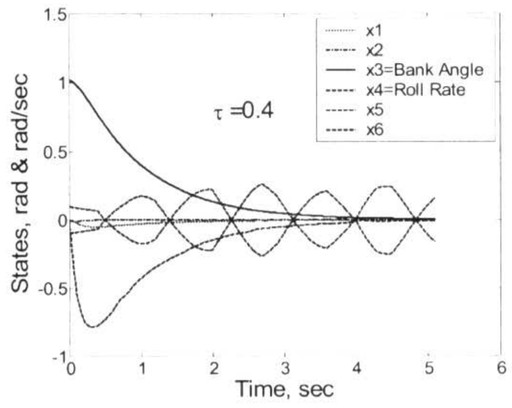


Fig.17. Time history of states for closed-loop MIMO system for $\tau = 0.4$

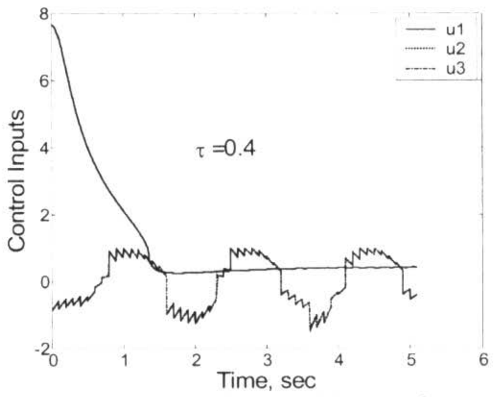


Fig.18. Time history of control inputs for $\tau = 0.4$

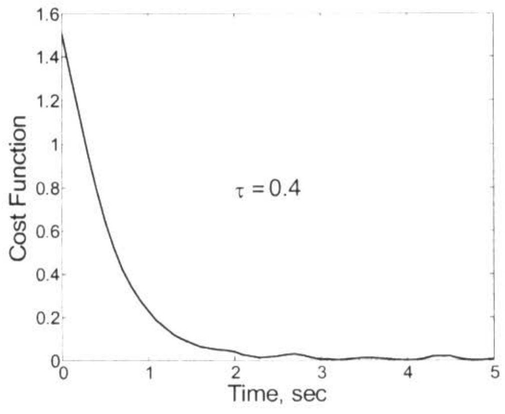


Fig.19. Time history of cost function for $\tau = 0.4$

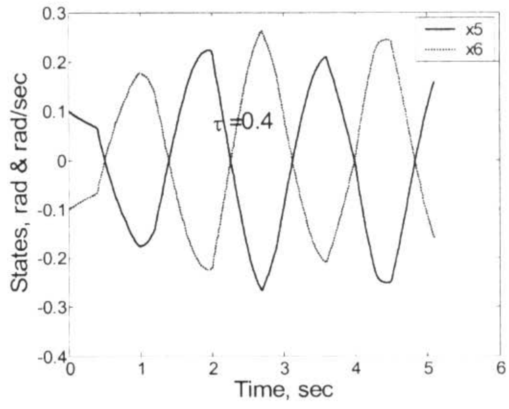


Fig.20. Time history of 5th and 6th states for $\tau = 0.4$

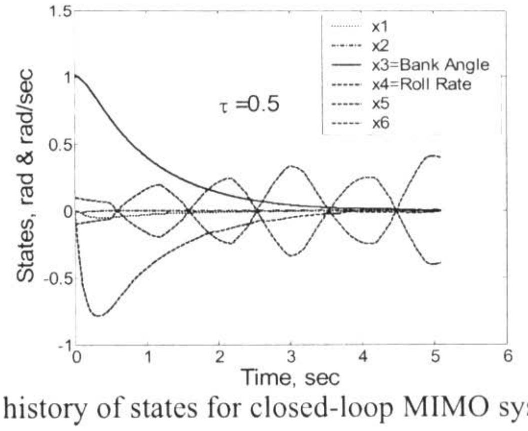


Fig.21. Time history of states for closed-loop MIMO system for $\tau = 0.5$

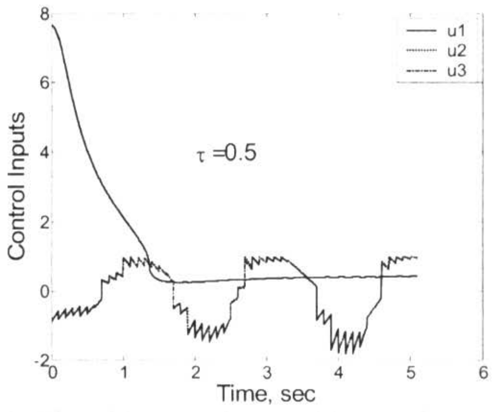


Fig.22. Time history of control inputs for $\tau = 0.5$

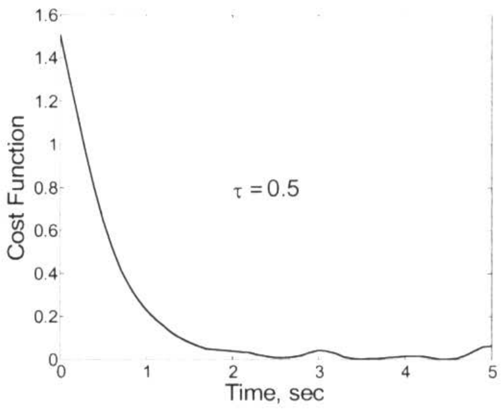


Fig.23. Time history of cost function for $\tau = 0.5$

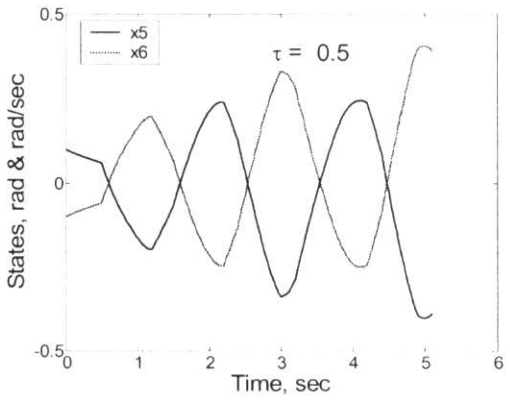


Fig.24. Time history of 5th and 6th states for $\tau = 0.5$

2.4 Conclusions

In this research, a quasi receding horizon control (RHC) is applied to the MIMO dynamics of vortex coupled delta wing aircraft in roll manoeuvre to study the effect of the time delay in SMA control inputs. Simulation results show that for small enough τ (time delay of SMA), RHC can handle the SMA's time delay; in fact, as τ goes up, the system becomes unstable. According to the simulation results, for $\tau \leq 0.25$ (time delay), RHC can control the system properly.

2.5 References

[1] Pakmehr M., Gordon B.W. and Rabbath C. A., "Control Oriented Modeling and Identification of Delta Wing Vortex-Coupled Roll Dynamics", to be presented at the American Control Conference 2005, Portland, Oregon, June 8-10, 2005 (to be published).

[2] M. Pakmehr, B. W. Gordon, "Innovative Methods of Flight Control by Manipulation of the Flow Structure" Internal report No.5, submitted to DRDC.

[3] X. Z. Huang, "Non-Linear Indicial Response and Internal State-Space (NIRISS) Representation and Its Application on Delta Wing Configurations", RTO Technical Report, RTO-TR-047.

[4] William Dunbar, "Distributed Receding Horizon Control of Multi agent Systems", PhD thesis, Control and Dynamical Systems, California Institute of Technology, April, 2004.

[5] M. B. Milam, R. Franz, J. E. Hauser, R. M. Murray, "Receding horizon control of a vectored thrust flight experiment", submitted to IEE Proceedings on Control Theory and Applications.

[6] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems", PhD Thesis, California Institute of Technology, Pasadena, CA, 2003.

CHAPTER 3: ROBUST QUASI RECEDING HORIZON CONTROL APPROACH FOR VORTEX BREAK DOWN AUGMENTED ROLL DYNAMICS OF DELTA WING AIRCRAFT

(Task # 4)

Hojjat A. Izadi
PhD Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory
Department of Mechanical and Industrial Engineering
Concordia University
Montréal, Québec, Canada H3G 1M8

October 2005

Abstract

This report documents the work related to task #4 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". In this report, Receding Horizon Control (RHC) method is used for the flow control problem of a delta wing aircraft which augments the "vortex coupled break down location" for the roll manoeuvre. Because of nonlinearities and uncertainties in the vortex coupled dynamics, the existing ordinary RHC methods can not be applied to this dynamics; therefore, it is required to develop a robust RHC scheme. Moreover, there is no design procedure to tune the parameters of the RHC controller systematically for nonlinear systems with uncertainty and large number of constraints on the states and inputs. In this research, based on a proposed robust quasi RHC scheme, a design procedure is proposed so that all parameters of the RHC controller can be tuned. The stability of the robust RHC controller is also proven. The effectiveness of the design procedure is evaluated by applying the robust RHC to both the 4th order and the MIMO dynamics of the "vortex coupled roll dynamics of delta wing aircraft" with delay, unmodeled uncertainty and disturbances. Simulation results show that the proposed robust RHC can stabilize the vortex coupled dynamics of delta wing with a remarkable performance. In all cases, the region of attraction of the robust RHC and also terminal region is obtained and discussed.

Nomenclature

A, B	system matrices	L	Lipchitz constant
b	wing span	K	controller gain
P,Q,R	Weight matrices of quadratic cost function	α	angle of attack
q	dynamic pressure	T	finite horizon time
I	moment of Inertia	ϕ	roll angle
X_{vb}	Non-dimensional vortex breakdown location	fc	friction coefficient
Xs	static term in X_{vb}	s	wing element area
Cl	roll moment coefficient	μ	Uncertainty coefficient

Subscripts

actual	actual system	*r*	right wing vortex
cont	Controller	*sma*	shape memory alloy
l	left wing vortex	*VB*	vortex breakdown
lpf	Low Pass Filter	*w*	wing

Abbreviations

RHC	Receding Horizon Control
SMA	Shape Memory Alloy
MIMO	Multi-Input Multi-Output
LQR	Linear Quadratic Regulator
RDDWAAVCBDL	Roll Dynamics of Delta Wing Aircraft Augmenting Vortex Coupled Break-Down Location

3.1 Introduction

This report addresses the following task related to the project "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques":

Task 4: Apply optimization-based receding-horizon control approaches to the flow control problem

The RHC method is based on a repeatedly on-line solution to a "finite horizon open loop optimal control" problem. An optimal control problem is solved for a period of time called the *prediction horizon*. Then, the first portion or the second portion of the computed optimal input may be applied to the plant in a period of time called the *execution horizon* until the next sampling of the states becomes available [1].

The RHC has traditionally found successful applications in the chemical process control, where the dynamics are slow. A very high computation burden of the RHC has prevented its application to nonlinear systems with fast dynamics like high performance aircrafts. However, with the advent of high speed computers and the development of the new computational methods in recent years, the RHC has been successfully applied to the nonlinear control problems with fast dynamics [2].

The "roll dynamics of delta wing aircraft augmenting a vortex coupled break-down location" (RDDWAAVCBDL) is a nonlinear dynamics containing state delay, uncertainty, friction and zero dynamics with a weak stability behaviour of the open loop system. It is obvious that the ordinary and typical existing RHCs could not be applied to such a problematic and complex dynamics; consequently, some revisions should be made to the ordinary RHC schemes.

The main idea in this report is to prove the robust stability of the Quasi RHC [3] for nonlinear systems with unmodeled dynamics. Then, based on this robust RHC scheme a design procedure will be developed so that all matrix penalties of cost function (*P, Q, and R*) and *final horizon time* (*T*) can be tuned for the uncertain systems.

The stability of the quasi RHC is proven in [3] for nonlinear systems without perturbation and any kind of uncertainty. The reason beyond the fact that the quasi RHC is used in this study is that it is easy to implement this kind of RHC to nonlinear systems. Also, the computation burden is less in comparison with other kinds of RHC schemes like dual-mode RHC [4]; because, in the quasi RHC it is not required to apply the terminal constraint. This terminal constraint is used to guarantee the stability of RHC in other kinds of RHC schemes and instead, some offline computations and investigations are done. In other words, the RHC controller parameters are tuned so that the closed-loop stability is guaranteed. Moreover, by the means of quasi RHC it is possible to discuss the region of attraction and also terminal region easily.

Only two papers addressed the robust stability of RHC. In the first one [5], authors used a H_∞ method to prove the robust stability of the RHC for systems with input disturbances; however, unmodeled dynamics and state uncertainty are not discussed in that paper. Also, in [4], the robust stability of a dual mode RHC controller for nonlinear systems is discussed theoretically; however, the practical and implementation issues are not discussed. It is important to mention that the dual mode RHC is not used in today's fast dynamics applications because of its difficulties and instead quasi RHC is being used due to its prominent benefits.

In the dual mode control, inside a terminal region a local feedback control law and, outside this terminal region an RHC is applied to the system. Whereas, in the quasi RHC, the RHC control input is applied to the system inside and outside the terminal region.

Some benefits of the RHC make it a good candidate to be used for the flow control problem of aerospace vehicles which are nonlinear, fast, constrained and uncertain dynamics. For instance, by the means of RHC, it is possible to design a feedback control law for nonlinear systems. Furthermore, since the control law is generated in a repeated fashion, it is possible to feedback the information through the computing of each optimal control law. This property makes the RHC flexible for control of systems with variable parameters and unknown environmental conditions. Excellent capability to handle the constraints is another unique advantage of the RHC. The optimality of the RHC is also a considerable advantage.

In this report, firstly the flow control problem is discussed; it is indicated that the RHC controller should fulfill some requirements to be applicable to flow control problem. Then, quasi RHC is introduced and the benefits of quadratic cost functions and flat outputs in reducing the computation burden are discussed. Secondly, some lemmas and theorems are presented to prove the robust stability of the RHC for uncertain systems. We will end up with a design procedure for nonlinear systems with unmodeled dynamics so that all controller parameters like penalties *P, Q, R*, and *finite prediction horizon* time (*T*) is tuned. After that, to verify the design procedure, a 2nd order numerical example is presented. Thirdly, this robust RHC is applied to the 4th order RDDWAAVCBDL (C_l and delay terms are considered as the unmodeled dynamics and nonlinearity correspondingly). To incorporate the shape memory alloy (SMA) actuators, the robust RHC is applied to MIMO of the RDDWAAVCBDL, which is a 6th order dynamics; this controller is also applied to the delta wing dynamics with new SMA antagonistic actuation system in Appendix E. Some numerical simulations are done to discuss the effectiveness and performance of the proposed robust quasi RHC scheme.

3.2 Receding horizon control of the flow control problem

Some features of the flow control problem have made it a challenging problem among the control designers. For instance, there are plenty of actuator saturations and constraints that must be handled. Also, all manoeuvres in flow control problem are very fast. Furthermore, there are some uncertainties in the model driving designers to do an uncertainty analysis. Some nonlinear behaviours such as state delay add also to the problem. Hence, an RHC has to be developed which can overcome all these difficulties; in fact, the RHC must:

1- Handle all saturations and constraints.
2- Be fast enough to handle the fast dynamics.
3- Be robust to unmodeled uncertainty and disturbances.
4- Handle nonlinearities like delay.

All these aspects are discussed in this report; in fact, we will design and formulate an RHC scheme that fulfills all of the above requirements.

Achieving the first requirement (handling the constraints) is quite simple because all formulations of the RHC could handle all kinds of constraints and saturations. However, by the means of the quasi RHC, this can be done easily because it is not required to check the final constraint in the quasi RHC scheme; this relaxes the final constraint and also can reduce the on-line computation burden which fulfills the second requirement as well. Also, by benefiting of the *flat outputs*, it is possible to reduce the dimension of the optimization problem and hence reduce the computation burden (Appendix A).

3.2.1 Quasi Receding Horizon Control

Suppose that the nominal model is presented as the following general form:

$$\dot{x} = f\big(x(t), u(t)\big) \qquad x(0) = x_0 \tag{1}$$

Where $\mathbf{x}(t) \in \Re^n$ is the state of the system and $\boldsymbol{u}(t) \in \Re^m$ is the input vector satisfying the constraints:

$$u(t) \in U \qquad \forall t \geq 0 \tag{2}$$

U is the set of allowable inputs. In order to meet the necessary conditions for stability; it is also assumed that:

A1- f is twice continuously differentiable.

A2- U is compact and convex and contains origin.

A3- system (1) has a unique solution for a given initial condition [1].

Then, under these circumstances the RHC is the repeated solution of the following problem [6]:

Problem 1: Find

$$\min_{\mathbf{u}(.)} \int_0^T q(x(\tau),u(\tau))d\tau + V(x(T)) \tag{3}$$

Subject to

$$\dot{x}(t) = f(x(t),u(t)) \qquad x(0) = x_0 \tag{4}$$

$$lb_0 \le \psi(x(0),u(0)) \le ub_0 \tag{5}$$

$$lb_t \le S(x(t),u(t)) \le ub_t \tag{6}$$

$$x(T) \in \Omega_r \tag{7}$$

Where

$$\Omega_r = \{x \in \Re^n : V(x) \le r\} \tag{8}$$

ψ is an initial constraint on the parameters and S is a trajectory constraint enforced over the entire time interval. lb and ub are the lower and upper bounds respectively on the constraints. Constraint (7) is added to guarantee the stability of the RHC; in fact, by the means of the RHC the trajectory of system is driven to a neighbourhood of the origin and after that a linear local feedback controller stabilizes the system; hence, for the open loop system, control input is as follows:

$$u(s) = \begin{cases} u^*(s;x(t),t) & x \in X \cap \Omega_r^c \\ Kx(s) & x \in \Omega_r \end{cases} \tag{9}$$

Where, X is the set of initial states that there exists a feasible solution for them, Ω_r^c is the set of states which do not belong to Ω_r and K is the gain of local feedback controller.

Remark: in the quasi RHC the local feedback controller is used in offline computations to prove the stability of the RHC scheme and design the cost function penalties; in that, in the online computations only the RHC controller scheme is used.

Repeating this computation yields a closed-loop feedback control law. For a receding horizon sampling period δ when δ belongs to (0, *T*]; then, the closed-loop system is represented as:

$$\begin{aligned} &\dot{x}(\tau) = f\left(x(\tau), u^*(\tau)\right) \\ &u^*(\tau) = u^*(\tau; x(t)) \qquad \tau \in [t, t+\delta), \quad 0 < \delta \le T \end{aligned} \tag{10}$$

Where $u^*(\tau; x(t)), \quad \tau \in [t, t+T]$, is the optimal control obtained from the above problem with the initial condition ***x****(t)*, and *t* is the start time of the optimization process and the instant at which states are sampled.

3.2.2 Selection of appropriate cost function to achieve good stability and performance properties

Solving the optimal **Problem 1** repeatedly leads to a closed-loop feedback control, but computationally solving an optimal control problem with cost function (3) is typically demanding and time consuming. Consequently, it is desired to select $q(x(0t, u(t))$ and *V(T)* such that the computational effort in the optimization is reduced and hence, the RHC can be used for fast flow dynamics. One potential choice is quadratic form functions due to their unique and excellent properties in optimization problems. In appendix B the advantages of quadratic cost functions in the optimization problems are discussed completely.

3.2.3 Quasi Receding Horizon Control with Quadratic Cost Function

With the following selections, the RHC problem becomes quadratic [7]:

$$q(x(t), u(t)) = \frac{1}{2} x^T(t) Q x(t) + \frac{1}{2} u^T(t) R u(t) \tag{11}$$

And:

$$V(x(T)) = \frac{1}{2}x^T Px \tag{12}$$

$Q,P \in \Re^{n\times n}$ and $R \in \Re^{m\times m}$ denote positive-definite, symmetric weighting matrices, T is a finite prediction time and $\boldsymbol{x}(t;\boldsymbol{x}_0)$ denotes the trajectory of the system (1) driven by $u(t)$ starting from the initial condition x_0.

3.2.4 Robust RHC Approach for Nonlinear Systems

To handle the uncertainties and disturbances, it is required to design a robust RHC scheme for nonlinear systems; this will fulfill the third and forth requirements of section 2. Suppose the nominal system (1) and the corresponding actual system:

$$\dot{x} = f_{actual}\left(x(t),u(t)\right) \qquad x(0) = x_0 \tag{13}$$

And consider the Jacobian linearization of the system (1) at origin is described by:

$$\dot{x} = Ax + Bu \qquad x(0) = x_0 \tag{14}$$

Where

$$A = \frac{\partial f}{\partial x}(0,0) \tag{15}$$

And

$$B = \frac{\partial f}{\partial u}(0,0) \tag{16}$$

And also, suppose that:

$$\left\| f_{actual}\left(x(t),u(t)\right) - f\left(x(t),u(t)\right) \right\| \le \mu \|x\| \tag{17}$$

To prove the robust stability of the RHC for uncertain system (13) with unmodeled uncertainty (17), we need the following three lemmas:

Lemma 1: there exists $r > 0$ such that Ω_r is an invariant region of attraction for nominal system (1) and actual system (13) by the feedback control input $u=kx$.

Proof: the existence of such region of attraction for nominal system without model error is proven in [3]; also, it is proven that if

A4- the Jacobian linearization system (14) is stabilizable

Then, the following Lyapanov equation has a unique solution:

$$(A_k + cI)^T P + P(A_k + cI) = -Q^* \tag{18}$$

Where, $A_k = A + BK$, $Q^* = Q + K^T RK$ and $c < -\lambda_{\max}(A_k)$; and, $\lambda_{\max}$ is the largest eigenvalue of A_k. It is assumed that the linear system (14) is stabilizing with control input $u=kx$; then, all eigenvalues of A_k are negative and $c < -\lambda_{\max}(A_k)$ guarantees that with positive definiteness of $Q^* = Q + K^T RK$, there exists a unique positive definite solution *P*, for Lyapanov equation (18).

To prove that Ω_r is an invariant region of attraction for actual system with local feedback control, first of all find a constant r_1 such that $kx \in U$ for all $x \in \Omega_{r_1}$; then, differentiate *V(x)* along the trajectory of actual system (13):

$$\frac{d(V(x))}{dt} = x^T P(f_{actual} - A_k x) + (f_{actual}{}^T - x^T A_k^T)Px + x^T(A_k{}^T P + PA_k)x \tag{19}$$

Suppose that:

$$\phi(x) = f - A_k x \tag{20}$$

Then:

$$\|f_{actual} - A_k x\| \le \|f_{actual} - f\| + \|f - A_k x\| \le \mu\|x\| + \|\phi(x)\| \tag{21}$$

Also suppose that:

$$\mu_\phi = Sup\{\frac{\|\phi(x)\|}{\|x\|} : x \in \Omega_{r_1}\} \tag{22}$$

Then, inequality (21) becomes:

$$\|f_{actual} - A_k x\| \le (\mu + \mu_\phi)\|x\| \tag{23}$$

Using norm, inequality (23) and using the fact that the norm of a matrix is greater than any of its eigenvalues, we will have:

$$x^T P(f_{actual} - A_k x) \le \|x\|\|P\|\| \ (f_{actual} - A_k x) \ \| \le \|P\|\|x\|^2 (\mu + \mu_\phi) \le \|P\| \frac{\mu + \mu_\phi}{\lambda(P)} . \|x\|_P^2 \tag{24}$$

If we choose *r* such that:

$$\mu + \mu_\phi \le c\frac{\lambda_{\max}(p)}{\|P\|} \tag{25}$$

Then, from (25) and (24), we will end up with:

$$x^T P(f_{actual} - A_k x) \le c.\|x\|_P^2 \tag{26}$$

The same derivation can be done for:

$$(f^T{}_{actual} - x^T A^T{}_k)Px \le c.\|x\|_P^2 \tag{27}$$

Putting (26) and (27) into (19) yields:

$$\frac{d(V(x))}{dt} = \frac{d(x^T Px)}{dt} \le - x^T Q^* x \tag{28}$$

Inequality (28) is obtained along the trajectory of actual systems (13) with control input *u=kx* in the set Ω_r and under the condition (25). Since *Q* is positive definite, *V(x)* is decreasing in Ω_r; this implies that any trajectory of actual system (13) which starts in Ω_r and is controlled by local feedback control *u=kx* remains in Ω_r; consequently, Ω_r is an invariant region of attraction for actual system (13) with local feedback control.

The next lemma discusses the *feasibility* of control input for the actual system with model error. By the *feasibility* of RHC we mean that there exists a valid control input (belongs to U) and a *finite horizon time* (*T*) so that the system (13) is driven to Ω_r at time *T*.

According to [8], *feasibility* of control input at time *t=0* for nominal system implies its *feasibility* for all *t > 0* for nominal system without model error. Hence, for actual system (13) with model error (17), it is enough to prove that there exists a *feasible* control at time *t=0*. The following lemma will discuss the conditions under which the existence of a *feasible* control at *t=0* is guaranteed for open loop system.

Lemma 2: for the actual system (13) with model error (17), there exists a valid *r >0* such that the actual system can be driven to Ω_r by the open loop receding horizon optimal control.

Proof: suppose that there exist a feasible solution to open loop *RH (t,x(t),u,T)* (starting at time *t*, with initial condition *x(t)*, and finite horizon time *T*) for nominal system with Ω_r; this means:

$$x(t+T) \in \Omega_r \Rightarrow x^T(t+T)Px(t+T) \le r \tag{29}$$

In the case of actual system, it is obvious that if the control input *u(s; x(t), t)* is applied to the actual system for $s \in [t, t+T]$ then the final trajectory *x(t+T)* may not enter Ω_r (figure 1) due to model error:

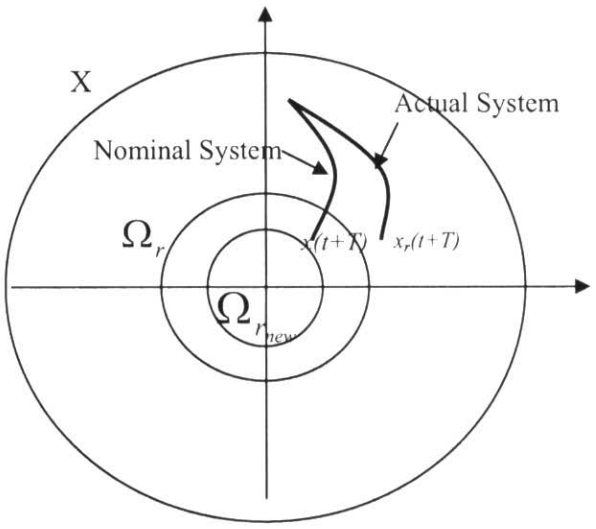


Fig. 1: Region of attraction and terminal region for both actual system and nominal system

In the above figure, $x_r(t+T)$ denotes the final state of actual system which is different from the final state of the nominal system and may not enter Ω_r because of model error. This implies that the terminal region Ω_r must be modified for the nominal system so that the terminal state of the actual system, $x_r(t+T)$, enters into Ω_r. Hence, we have to define a new r_{new} and a set $\Omega_{r_{new}}$ for nominal system so that the trajectory of actual system enters into Ω_r.

Suppose that b_{Ω_r} is the boundary of Ω_r and $b_{\Omega_{r_{new}}}$ is the boundary of $\Omega_{r_{new}}$; then, to find the set $\Omega_{r_{new}}$, we have to push the boundaries of Ω_r according to following inequality:

$$b_{\Omega_{r_{new}}} \leq b_{\Omega_r} - \left| x_r(t+T; x(t), t) - x(t+T; x(t), t) \right| \tag{30}$$

To find such a r_{new} so that the inequality (30) is satisfied, we can use the following analysis along with Bellman-Grownwall lemma and Variation Calculus, suppose that:

A5- both nominal and actual systems are *Lipchitz continuous* with *Lipchitz constant* L. Then, the following is a consequence of Belman-Grownwall lemma for nominal system (1) and actual system (13) with model error (17):

$$\|x_{actual}(t+T;x(t),t)-x(t+T,x(t),t)\|\leq\frac{\mu l}{L}(\exp(LT)-1) \quad (31)$$

Where, $\|x\|\leq l$ on set X (region of attraction of actual system with RHC).

Suppose the variation variables δx and δV , then:

$$V(x)=x^TPx\Rightarrow V(x)+\delta V(x)=x^TPx+x^TP\delta x+\delta x^TPx+\delta xP\delta x \quad (32)$$

Omitting non-variational terms yields:

$$\delta V(x)=x^TP\delta x+\delta x^TPx+\delta xP\delta x \quad (33)$$

The last term in the above equation is the product of two variation variables and is negligible; hence, taking norm yields:

$$\|\delta V(x)\|=2\|x^TP\delta x\|\leq 2\|x^TP\|\|x\| \quad (34)$$

The term $\|\delta x\|$ can be calculated from (31) which is a consequence of Bellman-Grownwall lemma. Then, $\Omega_{r_{new}}$ is defined as following:

$$\Omega_{r_{new}}=\left\{x\in\Re^n:V(x)\leq r-2\|x^TP\|\frac{\mu l}{L}(\exp(LT)-1)\right\} \quad (35)$$

Remark: if the model error be such that the right hand side of (35) be zero (or negative) it means that for this actual system the final terminal region set is empty ($\Omega_{r_{new}}\subset\phi$) and the inequality (7) converts to the final equality $x^*(t+T;x(t),t)=0$ which is difficult to be satisfied in online optimization.

Then, with the $\Omega_{r_{new}}$, it is obvious that for the nominal system $x_m^*(t+T;x(t),t)\in\Omega_{r_{new}}$ implies that $x_r^*(t+T;x(t),t)\in\Omega_r$ and according to lemma1 and since Ω_r is an invariant region of attraction for actual system, the actual system will be stabilized by the local feedback control $u=Kx$ and then $x_r^*(t+\delta+T;x(t+\delta),t+\delta)\in\Omega_r$.

To prove the stability of the closed loop system, we use the Lyapanov analysis. The quadratic performance index with terminal penalty is chosen as Lyapanov function candidate. The negative definiteness of this function for closed-loop system is approved in the following lemma.

Lemma 3: suppose that:

A6- There exists a *feasible* solution of the RHC problem for the actual system at time t that drives the final state of actual system at time $t+T$ to Ω_r (i.e. $x^*_{actual}(t+T;x(t),t)\in\Omega_r$). Also, suppose that the linerized system of nominal system (1) is stabilizing in Ω_r (A4); then, there exists a $\varepsilon > 0$ such that the following inequality holds for actual system:

$$\frac{\partial J(\tau;x(t),t)}{\partial t} \leq -\varepsilon \qquad ,\tau\in[t,t+\delta] \tag{36}$$

Proof: Appendix C.

Theorem1. Suppose that the assumptions A1-A6 are satisfied and model error is such that there exists a relation like (17) between nominal and actual system. Then, the closed-loop quasi receding horizon control of actual system with model error is asymptotically stable for sufficient small *execution time*.

Proof: we use the definitions of asymptotic stability in the sense of Lyapanov [9]:

Definition of *Asymptotic Stability*: let $x=0$ be an equilibrium point for (1). Suppose that $V(x):\Re^n \to \Re$ is a continuously differentiable function such that:

I. V(x) is positive definite

II. $\|x\| \to \infty \Rightarrow V(x) \to \infty$

III. $\dot{V}(x) < 0 \quad \forall x \neq 0$

Then, $x=0$ is globally asymptotically stable.

Suppose that:

$$V(x) = J^{*}_{actual}(x,t,t+T) \tag{37}$$

According to (11) and (12), $J^{*}_{actual}(x,t,t+T)$ is quadratic function. Also, *V(x)* is positive definite because *R, Q* and *P* are positive definite; condition II is straightforward for quadratic functions. Moreover, from lemma 3, III is satisfied and this completes the proof.

3.2.5 Design Procedure:

According to previous lemmas and theorem, in this section a design procedure is developed by the means of which parameters of the robust RHC controller is tuned so that the stability of the closed-loop system is guaranteed. It is assumed that the assumptions A1-A6 hold.

Step1. Define the constraints and limitations on the states and inputs.

Step2. Obtain the Jacobian linearized form of nominal nonlinear system (*A, B*), and check whether the linerized system is stabilizable or not?

Step3. Choose positive definite *Q* and *R* for penalties of quadratic performance index.

Step4. By the means of one of linear feedback control methods like LQR obtain a linear feedback gain *K* for linearized system

Step5. Choose *c* so that $c < -\lambda_{\max}(A_k)$ and $A_k = A + BK$.

Step6. Solve Lyapanov equation (18) and obtain *P*.

Step7. Find the largest possible r_1 such that: $\forall x \in \Omega_{r_1} \Rightarrow K.x \in U$, (U is the set of admissible controls).

Step8. Find the largest possible $r \in [0, r_1]$ such that inequality (25) holds.

Step9. Choose *T* (*Finite Horizon time*) and find the largest possible $r_{new} \in [0, r]$ according to (35) or any other methods.

Step10. Check if with the designed parameters (*P, Q, R, T*), there is a feasible open loop receding horizon optimal control for nominal system or not? (i.e. $x^{*}_{m}(t+T; x(t), t) \in \Omega_{r_{new}}$). If not, either reduce the boundaries of X (region of attraction of RHC) or go to step 3 and redefine *Q, R, T* and other design parameters.

3.2.6 Numerical Example:

The following numerical example will verify the results. Suppose the following actual nonlinear system:

$$\begin{cases} \dot{x}_1(t) = x_2(t) + \mu_1 x_2(t) \\ \dot{x}_2(t) = -x_1(t) - 2x_2(t) + u(t)(2x_2(t)+1) + \mu_2 x_2(t) \end{cases} \tag{38}$$

Where terms multiplied by μ_1 and μ_2 are unmodeled dynamics:

Step1. System is under the following limitations:

$$\begin{cases} -1 \le x \le 1 \\ -1 \le u \le 1 \end{cases} \tag{39}$$

Step2. The linearization of this system around equilibrium point (0, 0) yields:

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -x_1(t) - 2x_2(t) + u(t) \end{cases} \tag{40}$$

Both eigenvalues of linearized system are -1; then the linearized system is stabilizable.

Step3. Choose:

$$Q = R = I \tag{41}$$

Step4. By the means of LQR:

$$K = [-0.4142 \quad -0.4142] \tag{42}$$

Step5.

$$c = 0.95 \tag{43}$$

Step6.

$$P = \begin{bmatrix} 174.7573 & 117.8077 \\ 117.8077 & 80.8581 \end{bmatrix} \tag{44}$$

Step7. Under the conditions (39), and with K:

$$r_1 = 491.2308 \tag{45}$$

Step8. Choosing $\mu = 0.3$, then:

$$r = 195.1239 \tag{46}$$

Step9. Choosing *l=0.5, L=1, T=2 sec,*

$$r_{new} = 3.0792 \tag{47}$$

Step10. In figure 2 these terminal regions are shown. Also, for some initial conditions the open loop optimal control of problem 1 is applied to both nominal and actual system and trajectories are depicted. The solid lines show the trajectories of actual system and the dashed lines show the trajectories of nominal system. As it is shown all trajectories of nominal system enter into $\Omega_{r_{new}}$ and all trajectories of actual system enter into Ω_r; this implies that the robust RHC controller with design parameters can stabilize the close loop system.

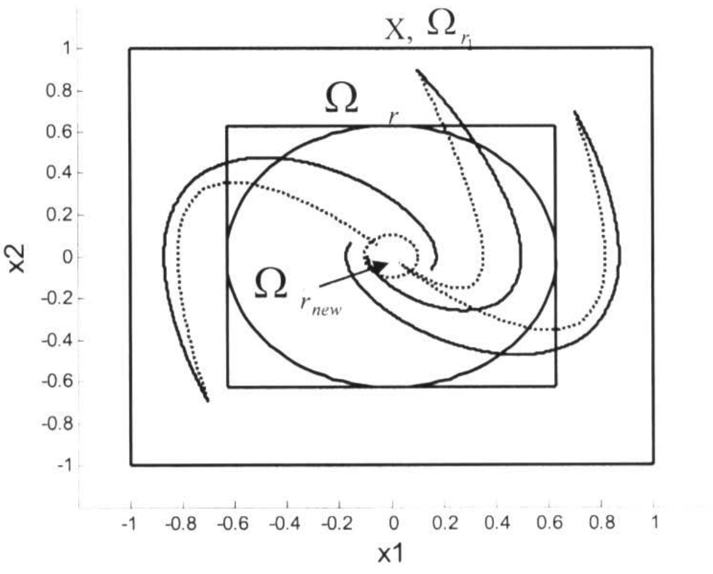


Fig.2. Terminal region for actual system and nominal system and trajectories of both actual and nominal system subject to open loop optimal control

3.3 Application to Vortex Coupled Dynamics

In this section, the robust quasi receding horizon control (RHC) discussed in the previous section is applied to the vortex coupled dynamics of a delta wing aircraft in roll manoeuvre as a sophisticated flow control problem. We start with some discussions on the vortex coupled dynamics; it is required to discuss how the delay term and the uncertainties of the vortex coupled dynamics are treated and handled in the robust RHC design.

3.3.1 The time delay terms of the vortex dynamics

One of the features of the flow control problem which is supposed to be handled by the means of robust RHC scheme is the state delay term. To achieve this objective, the delay term is regarded as a nonlinear term. Hence, since the designed robust RHC scheme can handle the nonlinearities, it can handle the delay term simply. According to step 2 of design procedure, this delay term will be linearized as we will see later and the difference between the nonlinear (delay) term and linear term is counted in the design procedure of robust RHC design. Actually, this is the advantage of the designed robust RHC controller that any kind of nonlinearity can be approximated by a linear term and then the difference between the nonlinear term and the corresponding linearized is counted in the design procedure to reduce the approximation error.

3.3.2 Robustness to unmodeled dynamics uncertainty and disturbances

Another important feature of flow control problem is unmodeled dynamics uncertainty and disturbances that must be analyzed in the design procedure. The unmodeled dynamics in the vortex coupled problem of delta wing is lift coefficient C_l; this can be done by the step 8 and 9 of the design procedure. This is discussed thoroughly in sections 3.4.1 and 3.5.1 correspondingly for 4th order and MIMO dynamics.

3.3.3 Reduction of computation burden:

Since the RHC is a solution of an optimal control problem which is done online; then, always a delay is induced in the on-line applications known as computation delay. This computation delay can affect the stability and performance. To reduce the computation delay, it is possible to reduce the computation burden. In our proposed scheme, this is done in two ways: firstly, by choosing the quasi RHC scheme; since the final constraint in this scheme is relaxed, this could reduce the computation burden. Secondly, by reducing the *final finite horizon time (T)* which is done in the step 9 of design procedure. In step 9, it is possible to tune *T for* better performance; the smaller *T*, the less computation burdens (the less computational delay).

3.3.4 Application to 4th order Vortex coupled Delta Wing:

The roll dynamics of delta wing aircraft augmenting vortex coupled dynamics is described by [10]:

$$\begin{cases} \dot{x}_1(t) = c\,x_2(t) \\ \dot{x}_2(t) = -c\,x_1(t) - \varepsilon_d\, x_2(t) + x_4(t) + x_4(t-\tau) \\ \dot{x}_3(t) = -\varepsilon_d\, x_3(t) + x_4(t) \\ \dot{x}_4(t) = u(t)/I_w - Cl(Xvb(x(t)))Q - f_c\, x_4(t) \end{cases} \tag{48}$$

The rolling moment coefficient C_l is a nonlinear function of vortex breakdown locations [11]; we assume it is a 3rd order polynomial in the following form:

$$C_l(X_{vbl}, X_{vbr}) = e_0 + e_1(X_{vbl} - X_{vbr}) + e_2(X_{vbl}^{\,2} - X_{vbr}^{\,2}) + e_3(X_{vbl}^{\,3} - X_{vbr}^{\,3}) \tag{49}$$

Where, X_{vbl} and X_{vbr} represent the vortex breakdown locations for the left and right vortices, and e_0, e_1, e_2 and e_3 are constant coefficients obtained using 3rd order least square curve fitting of the experimental data. Vortex breakdown locations are considered as following [11]:

$$X_{vbl}(t) = X_{sl}(t) + X_{sl}\, k_q(t)\, x_2(t) \tag{50}$$

$$X_{vbr}(t) = X_{sr}(t) + X_{sr}\, k_q(t)\, x_2(t) \tag{51}$$

Where, X_{sl} and X_{sr} are static components of vortex breakdown locations in the overall vortex breakdown locations [12], and:

$$k_q(t) = 0.91/\tan(\alpha(t)) \tag{52}$$

Where, angle of attack (α) is computed as follows:

$$\alpha(t) = \operatorname{atan}(\cos(\,x_1(t)) \cdot \tan(\,\sigma\,)) \tag{53}$$

And, bevel angle of the wing (σ) is an experimentally obtained value [12]. It is obvious that by considering (49), dynamics equation (48) becomes highly nonlinear; it is desired to design a robust RHC for this nonlinear dynamics; the robust RHC controller then can behave like a feedback controller for this nonlinear system.

3.4.1. Robust RHC Design for 4th order:

Equation (48) describes the vortex break down roll dynamics of delta wing aircraft; in this dynamics C_l is regarded as an unmodeled dynamics.

Step1. For a delta wing aircraft the following bounds are considered on the bank angle, roll rate and control input:

$$\begin{cases} -100^{\circ} \le \phi \le 100^{\circ} \\ -100 \text{ deg/sec} \le \dot{\phi} \le 100 \quad \text{deg/sec} \\ -10 \le u \le 10 \end{cases} \tag{54}$$

Then the following bounds can be considered on the states and control input, these are selected from the simulation results of [11]:

$$\begin{cases} -0.4 \le x_1 \le 0.4 \\ -0.4 \le x_2 \le 0.4 \\ -1.74 \le x_3 \le 1.74 \ (Rad.) \\ -1.74 \le x_4 \le 1.74 \ (Rad.) \\ -10 \le u \le 10 \end{cases} \tag{55}$$

Step2. Since the origin is the equilibrium of system then the linearized system is:

$$A = \begin{bmatrix} 0 & c & 0 & 0 \\ -c & -\varepsilon_d & 0 & 2 \\ 0 & 0 & -\varepsilon_d & 1 \\ 0 & 0 & 0 & -f_c \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ h_1 / I_w \end{bmatrix} \tag{56}$$

It is important to notice that the delay term is approximated by a non-delay term; choosing above parameters according to [11], we have:

$$\begin{cases} c = 31.4159 \\ \varepsilon_d = 0.1 \\ I_w = 0.0305 \\ f_c = 0.4 \\ h_1 = 1 \\ \overline{Q} = 27658.97 \end{cases} \tag{57}$$

Then:

$$A = \begin{bmatrix} 0 & 31.4159 & 0 & 0 \\ -31.4159 & -0.1000 & 0 & 2.0000 \\ 0 & 0 & -0.1000 & 1.0000 \\ 0 & 0 & 0 & -0.4000 \end{bmatrix} \qquad (58)$$

$$B = [\ 0;\ 0;0;\ 32.7761]$$

The eigenvalues of linerized system are:

$$\begin{cases} -0.0500 + 31.4159i \\ -0.0500 - 31.4159i \\ -0.1000 \\ -0.4000 \end{cases} \qquad (59)$$

All eigenvalues have negative real part; then, the linearized system is stabilizable.

Step3.

$$Q = R = I \qquad (60)$$

Where I is identity matrix

Step4. By the means of LQR:

$$K = [0.8700 \quad -1.0250 \quad -0.9019 \quad -1.0742] \qquad (61)$$

Step5.

$$Max(\lambda_{A_K}) = -1.0214 + 31.4317\text{i} \Rightarrow$$
$$c = 0.9 \qquad (62)$$

Step6. Solving Lyapanov equation for selected parameters above yields:

$$P = \begin{bmatrix} 8.0787 & 0.0540 & 0.5265 & -0.2271 \\ 0.0540 & 8.5553 & 0.4820 & 0.2856 \\ 0.5265 & 0.4820 & 9.6056 & 0.2916 \\ -0.2271 & 0.2856 & 0.2916 & 0.0559 \end{bmatrix} \qquad (63)$$

Step7. For new local controller to satisfy the input bounds (55), since U is large enough this can not reduce the terminal region; in fact, in the above region the local feedback controller with gain of step 4 is admissible; hence, with this new region of attraction:

$$r_1 = 35.1806 \qquad (64)$$

Step8. In this step, it is required to obtain the uncertainty parameter (μ). In the delta wing model, uncertainty consists of term $C_l(x_{vb})$; simulation results show that:

$$Max(C_l(x_{vb})) = 0.01 \tag{65}$$

That barely happens and typically is about 0.001. For region of attraction (55), we have:

$$\|x\| \leq 4.28 \tag{66}$$

Then:

$$\mu = \frac{Max(C_l)}{\|x\|} = 0.0023 \tag{67}$$

However, for some other perturbations which are negligible in comparison with this unmodeled uncertainty and to avoid selecting a conservative bound on the uncertainties, we choose:

$$\mu = 0.003 \tag{68}$$

Considering the delay term as the nonlinearity, μ_ϕ is computed as following:

$$\mu_\phi = Max(\frac{\|x_4\|}{\|x\|}) = 0.4 \tag{69}$$

Since in our case the nonlinearity is weak, no considerable changes happen for r:

$$r \cong r_1 = 35 \tag{70}$$

Step9. It is desired to have a smaller final horizon time T; because the smaller T, the less computation burden. Since the optimization time is done on-line, a small decrease in T can reduce the computation burden dramatically; then, set $T=1$ sec and $L=6$ and:

$$l = Max\|x\|_\infty = 1.74 \tag{71}$$

Hence:

$$r_{new} = 13.7070 \tag{72}$$

Step10. Figure 3 shows the trajectory of open loop nominal system with the design parameters above and with some initial conditions in the region of attraction; the final trajectory of system reaches:

$$x(t+T;\, x(t),\, t) = [\ -0.0395 \quad 0.0009 \quad 0.4647 \quad -0.6461\ ] \tag{73}$$

Thus:

$$x^T(t+T)Px(t+T) = 1.9043 < r_{new} \tag{74}$$

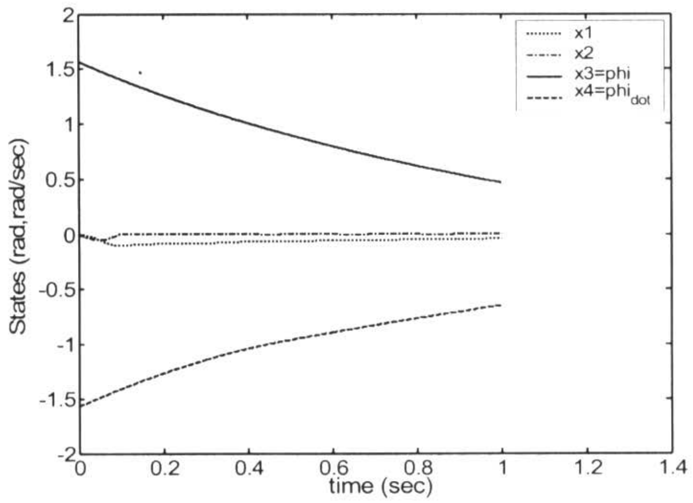


Fig3. Open loop trajectory of 4th order system

Hence, the final trajectory with open loop enters $\Omega_{r_{new}}$ that guarantees the fact that the final trajectory of actual system enters Ω_r. This implies that this control problem has a feasible solution; consequently, the closed-loop RHC with designed parameters is stable for actual system.

3.4.2. Closed-Loop RHC Design for 4th order roll dynamics:

To design RHC for vortex coupled delta wing above, consider the following flat outputs:

$$\begin{cases} z_1 = x_1 \\ z_2 = x_3 \end{cases} \tag{75}$$

Then, all states and inputs of system can be recovered as a function of above flat outputs and their derivatives:

$$\begin{cases} x_1 = z_1 \\ x_2 = \dot{z}_1 / c \\ x_3 = z_2 \\ x_4 = \dot{z}_2 + \varepsilon_d z_2 \\ u_1 = I_w[(\ddot{z}_2 + \varepsilon_d \dot{z}_2) + f_c(\dot{z}_2 + \varepsilon_d z_2) + C_l.\overline{Q}] \end{cases} \tag{76}$$

Then, with designed parameters in previous section (*P, Q, R* and *T*) which guarantee the robust stability of uncertain delta wing vortex coupled model, the robust quasi RHC is applied to the actual delta wing dynamics under constraints (55). Also, to study the robustness of the controller to the disturbances, a noise with magnitude of 10% of the amount of states is added to the actual model.

Matlab software has been used to simulate the system. Polynomial class functions have been used to estimate the flat outputs; the order of polynomials is three with second-order continuity at break points. A sequential quadratic programming (SQP) [13] method has been used to solve the optimal control problem. Briefly, in SQP method an optimization method, in which the cost function is quadratic, is solved by the means of any recursive method like *Steepest Decent* (SD) method. In each optimization, the previous optimized value of optimization parameters is used as the initial guess. The time history of states is depicted in figure 4 for an initial condition [0, 0, 58(deg), 0]. The execution time is $\delta = 0.1$.

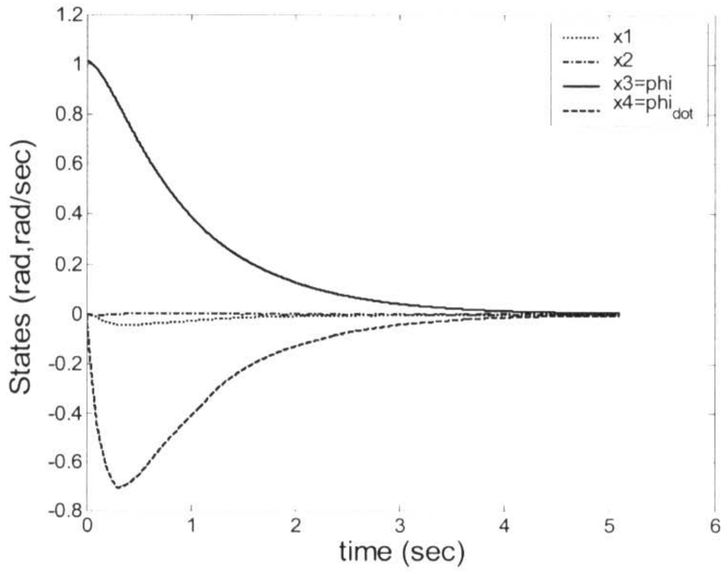


Fig.4. time history of states for closed-loop 4th order system

And the input profile is depicted in figure 5:

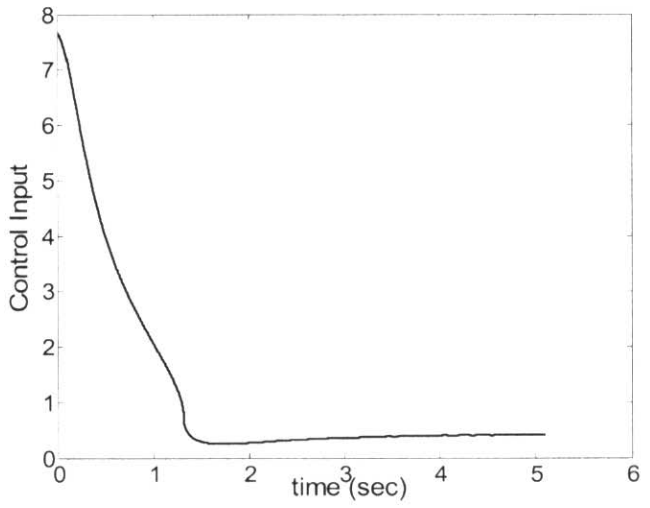


Fig.5. Time history of input for closed loop system

As it is shown, neither states nor input exceed the limitations and constraints (55) and system has a good performance with designed parameters. The lift coefficient which is uncertain unmodeled dynamics is depicted in figure 6.

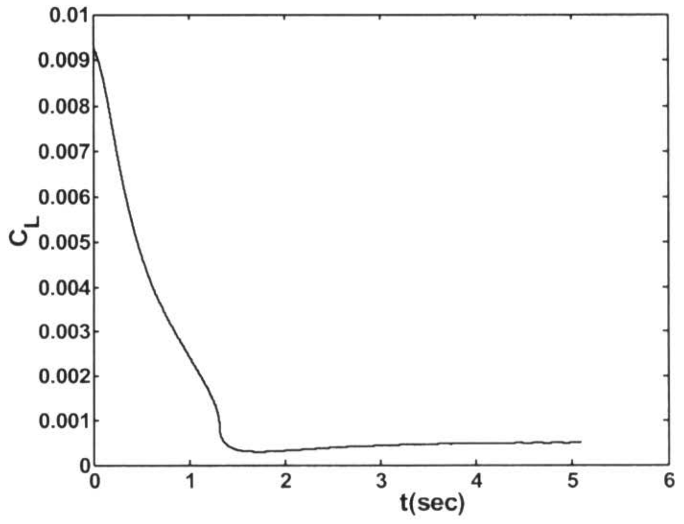


Fig6. Time history of C_l

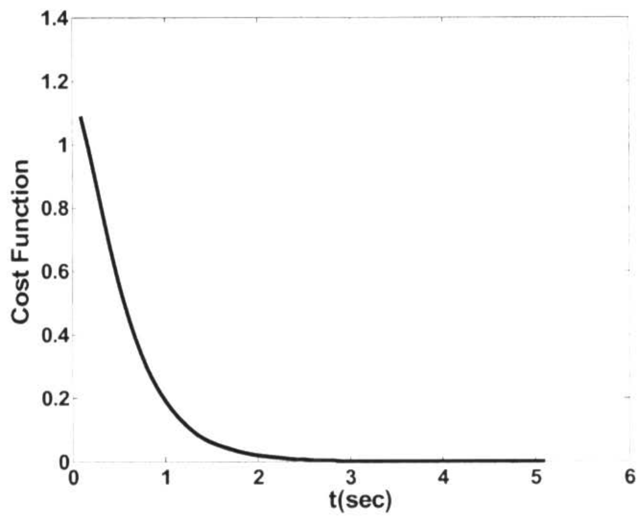


Fig.7. Time history of cost function

Time history of cost function is depicted in figure 7; as it is shown the cost function is decreasing which guarantee the stability of the RHC.

3.3.5 Application to MIMO Dynamics of Vortex coupled Delta Wing:

The dynamics of vortex coupled dynamics is described by:

$$\begin{cases} \dot{x}_1(t) = c\,x_2(t) \\ \dot{x}_2(t) = -c\,x_1(t) - \varepsilon_d\,x_2(t) + x_4(t) + x_4(t-\tau) \\ \dot{x}_3(t) = -\varepsilon_d\,x_3(t) + x_4(t) \\ \dot{x}_4(t) = u(t)/I_w - Cl(Xvb(x(t)))Q - f_c\,x_4(t) \end{cases} \tag{77}$$

To incorporate the dynamics of shape memory alloy (SMA) actuators the following filter-like equations are added to the system dynamics [10]:

$$\begin{cases} \dot{x}_5(t) = (-x_5(t) + h_2\rho_{vbl}u_2(t))/\varepsilon_l \\ \dot{x}_6(t) = (-x_6(t) + h_3\rho_{vbr}u_3(t))/\varepsilon_r \end{cases} \tag{78}$$

Equations (77) along with (78) are called the MIMO dynamics of roll dynamics of delta wing aircraft augmenting the vortex coupled break down.

3.5.1. Robust RHC Design for MIMO Roll Dynamics:

Again a robust RHC is designed for MIMO dynamics (Appendix D); like 4th order, C_l is considered as an unmodeled dynamics, all parameters of dynamics are selected like 4th order case as well.

3.5.2. Closed-Loop RHC Design for MIMO Roll Dynamics:

To design RHC for vortex coupled delta wing above, consider the following flat outputs:

$$\begin{cases} z_1 = x_1 \\ z_2 = x_3 \\ z_3 = x_5 \\ z_4 = x_6 \end{cases} \tag{79}$$

Then, all states and inputs of the system can be recovered as a function of above flat outputs and their derivatives:

$$\begin{cases} x_1 = z_1 \\ x_2 = \dot{z}_1 / c \\ x_3 = z_2 \\ x_4 = \dot{z}_2 + \varepsilon_d z_2 \\ x_5 = z_3 \\ x_6 = z_4 \\ \\ u_1 = I_w[(\ddot{z}_2 + \varepsilon_d \dot{z}_2) + f_c(\dot{z}_2 + \varepsilon_d z_2) + C_l.\overline{Q}] \\ u_2 = \dfrac{\varepsilon_{SMA}}{h_2 \rho_L}(\dot{z}_3 + z_3) \\ u_3 = -\dfrac{\varepsilon_{SMA}}{h_3 \rho_r}(\dot{z}_4 + z_4) \end{cases} \tag{80}$$

Then, with the designed parameters of appendix D, and the execution time, $\delta = 0.1$, the quasi robust RHC is applied to the actual vortex coupled delta wing model with all its nonlinearities including delay and uncertainties in lift coefficient. Also, to study the robustness of the controller to the disturbances a noise with magnitude of 10% of the amount of states is added to the actual model. Again, similar to 4th order, polynomial class functions have been used to estimate the flat outputs; the order of polynomials is three with second-order continuity at break points. The time history of states is depicted in figure 8. Also, time history of control inputs are depicted in figures 9 and 10. Like 4th order, the performance of MIMO system is good and the robust quasi RHC could stabilize the system properly.

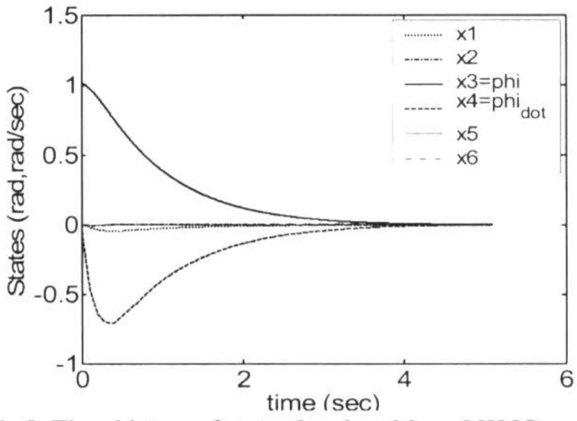


Fig.8. Time history of states for closed-loop MIMO system

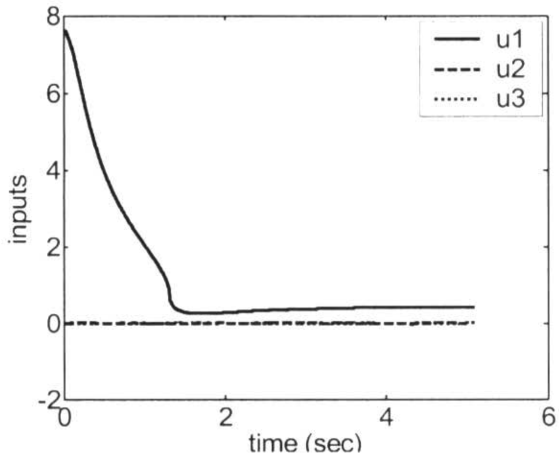


Fig9. Time history of inputs for closed-loop MIMO system

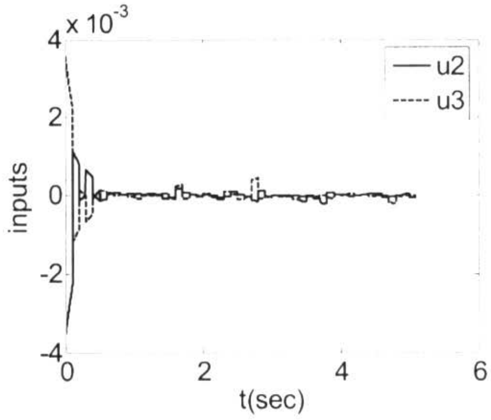


Fig10. Inputs to SMA

3.4 Conclusion

In this research, a robust quasi receding horizon control (RHC) is proposed and the stability of that is proven. Based on this robust RHC, a design procedure is developed that enables designers to tune the RHC parameters systematically. The proposed robust RHC scheme is applied to both 4th order and MIMO dynamics of vortex coupled delta wing aircraft in roll manoeuvre with time delay, unmodeled uncertainty and disturbances. Excellent simulation results show that the proposed robust RHC scheme can be used for flow control problems efficiently. Furthermore, it is shown that the controller can handle all the constraints and saturations. In addition, in the proposed controller, some of the computations are done off-line to meet the stability requirements and it is not required to fulfill the final constraint in on-line computations; hence, this can reduce the computation burden and make it possible to apply this controller to fast dynamics of flow control problem.

3.5 Appendix A: Flat Outputs

The dimension of the optimal control problem is reduced to a lower dimension that easing the formulation and reducing the amount of required computations for optimization process by means of so-called flat outputs. For applying the RHC method to the roll dynamics and solving the open loop optimal control problem the flat output advantages has been utilized. System (1) is called a flat system if there exist outputs z [7]:

$$Z=g(x,u) \tag{81}$$

Such that the states and the control signal can be recovered from z and its derivatives; that is,

$$(x,u)=h(z,\dot{z},....,z^{(r)}) \tag{82}$$

Equation (82) is only required to hold locally [7]. For a system being flat, it is required that all system behaviours including states and control inputs can be recovered from a finite number of flat outputs derivatives and without integration by the flat outputs; hence, h has to be a smooth function. The interested readers are referred to [7] for more details. If the dimension of z is smaller than that of x and u, the optimization problem is mapped to a lower dimension.

3.6 Appendix B: Quadratic Cost Functions

Suppose that the quadratic cost function is selected as:

$$\Phi(x) = C^T x + \frac{1}{2} x^T G x \tag{83}$$

Since G is the Hessian of Φ then, it is desired that all eigenvalues of G be positive for a unique global minimum; if some eigenvalues of G are positive and some are negative; then it means there are some local weak minimum points that may prevent to drive Φ to its global minimum point; in the case of all negative eigenvalues for G; there is no minimum point [14].
From Taylor series expansion:

$$\Phi(\hat{x} + \alpha P) = \Phi(\hat{x}) + \alpha P^T (G\hat{x} + C) + \frac{1}{2}\alpha^2 P^T G P \tag{84}$$

So, for optimality [14]:

$$Gx^* = -c \tag{85}$$

Where, x^* is the minimum point of Φ. If $\mathrm{u}_\mathrm{j} = \mathrm{P}$ is the *jth* eigenvector of G (Hessian matrix of Φ) then u_j is an orthonormal vector; hence, $\mathrm{u}_\mathrm{j} * \mathrm{u}_\mathrm{j} = 1$; also:

$$Gu_j = \lambda_j u_j \tag{86}$$

Then, from (84), (85) and (86), the following holds:

$$\Phi(x^* + \alpha u_j) = \Phi(x^*) + \frac{1}{2}\alpha^2 \lambda_j \tag{87}$$

So, the changes in Φ when moving away from x^* along the direction u_j depend on the sign of λ_j:

If λ_j is positive; then, Φ increases, it means that x^* is the minimum of Φ.

If λ_j is negative; then, Φ decreases.

If λ_j is zero; then, the value of Φ remains constant.

Also as mentioned above, when all eigenvalues of G are positive then x^* is the unique global minimum of Φ; hence, the quadratic cost function will be designed in such a way that Hessian matrix be positive definite; consequently, all of its eigenvalues are positive; and it is concluded that quadratic cost functions are suitable functions in optimal problems and this suggests quadratic functions are good candidates for cost functions.

3.7 Appendix C: Proof of Lemma 3

Lemma 3: suppose that

A6- There exists a solution to the RHC problem for actual system at time t that drives the final state of actual system at time $t+T$ to Ω_r (i.e. $x^*_{actual}(t+T;x(t),t)\in\Omega_r$), and also suppose that the linerized system of nominal system (1) is stabilizing in Ω_r (A4); then, there exists a $\varepsilon>0$ such that the following inequality holds for actual system:

$$\frac{\partial J(\tau;x(t),t)}{\partial t}\leq-\varepsilon \qquad ,\tau\in[t,t+\delta] \tag{88}$$

Proof: suppose that for the actual system:

$$J^*_{actual}(x(t),t,t+T)=\int_t^{t+T}(\|x_{actual}(s;x(t),t)\|_Q^2+\|u(s;x(t),t)\|_R^2)ds+\|x_{actual}(t+T;x(t),t)\|_P^2 \tag{89}$$

Then, for $\Delta t\in[t,t+\delta]$:

$$\begin{aligned}J^*_{actual}(x(t+\Delta t),t+\Delta t,t+\Delta t+T)=&\int_{t+\Delta t}^{t+\Delta t+T}(\|x^*_{actual}(s;x(t+\Delta t),t+\Delta t)\|_Q^2+\|u(s;x(t+\Delta t),t+\Delta t)\|_R^2)ds\\&+\|x^*_{actual}(t+\Delta t+T;x(t+\Delta t),t+\Delta t)\|_P^2\end{aligned} \tag{90}$$

Hence,

$$\begin{aligned}J^*_{actual}(x(t+\Delta t),t+\Delta t,t+\Delta t+T)=&\int_t^{t+T}(\|x^*_{actual}(s;x(t+\Delta t),t+\Delta t)\|_Q^2+\|u(s;x(t+\Delta t),t+\Delta t)\|_R^2)ds\\&-\int_t^{t+\Delta t}(\|x^*_{actual}(s;x(t+\Delta t),t+\Delta t)\|_Q^2+\|u(s;x(t+\Delta t),t+\Delta t)\|_R^2)ds\\&+\int_{t+T}^{t+\Delta t+T}(\|x^*_{actual}(s;x(t+\Delta t),t+\Delta t)\|_Q^2+\|u(s;x(t+\Delta t),t+\Delta t)\|_R^2)ds\\&+\|x^*_{actual}(t+\Delta t+T;x(t+\Delta t),t+\Delta t)\|_P^2\end{aligned} \tag{91}$$

Consider the following control input over the interval $[t,t+\Delta t+T]$:

$$u(s)=\begin{cases}u^*(s;x(t),t) & s\in[t,t+T]\\ K\,x(s) & s\in[t+T,t+\Delta t+T]\end{cases} \tag{92}$$

Where, K is the gain of local feedback controller. Since according to Lemma 2 at time T the trajectory of actual system enters Ω_r and inside this set, according to lemma 1, the feedback controller stabilizes the system then the above controller is valid; feasible and stabilizing for actual system.

Integrating of (28) over the interval $[t+T, t+\Delta t+T]$ yields:

$$\int_{t+T}^{t+\Delta t+T} \frac{d(x^T P x)}{dt} \le \int_{t+T}^{t+\Delta t+T} x^T Q^* x \Rightarrow \|x(t+T+\Delta t)\|_P^2 - \|x(t+T)\|_P^2 \le -\int_{t+T}^{t+T+\Delta t} (\|x(s)\|_Q^2 + \|u(s)\|_R^2) ds \quad (93)$$

Using (91) and (93) the following holds:

$$J^*_{actual}(x(t+\Delta t), t+\Delta t, t+\Delta t+T) \le J^*_{actual}(x(t), t, t+T) - \int_t^{t+\Delta t} (\|x^*_{actual}(s; x(t+\Delta t), t+\Delta t)\|_Q^2 + \|u(s; x(t+\Delta t), t+\Delta t)\|_R^2) ds \quad (94)$$

Dividing by Δt and taking limit:

$$\lim_{\Delta t \to 0} \frac{J^*_{actual}(x(t+\Delta t), t+\Delta t, t+\Delta t+T) - J^*_{actual}(x(t), t, t+T)}{\Delta t} \le - \lim_{\Delta t \to 0} \frac{\int_t^{t+\Delta t} (\|x^*_{actual}(s; x(t+\Delta t), t+\Delta t)\|_Q^2 + \|u(s; x(t+\Delta t), t+\Delta t)\|_R^2) ds}{\Delta t} \quad (95)$$

$$\frac{\partial J_{actual}(\tau; x(t), t)}{\partial t} \le -(\|x^*_{actual}(\tau; x(t), t)\|_Q^2 + \|u(\tau; x(t), t)\|_R^2) \quad (96)$$

And this completes the proof.

3.8 Appendix D: Robust RHC design for MIMO vortex coupled dynamics:

Step1. The constraints and limitations on states and control inputs are defined like 4th order and for the 5th and 6th states and controls we assume the following limitations:

$$\begin{cases} -1 \le x_5 \le 1 \\ -1 \le x_6 \le 1 \\ -10 \le u_1 \le 10 \\ -1 \le u_2 \le 1 \\ -1 \le u_3 \le 1 \end{cases} \tag{97}$$

Step2. Since the origin is the equilibrium of system then the linearized system becomes:

$$A = \begin{bmatrix} 0 & c & 0 & 0 & 0 & 0 \\ -c & -\varepsilon_d & 0 & 2 & 0 & 0 \\ 0 & 0 & -\varepsilon_d & 1 & 0 & 0 \\ 0 & 0 & 0 & -f_c & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ h_1 / I_w & 0 & 0 \\ 0 & \dfrac{h_2 \rho_{vbl}}{\varepsilon_{SMA}} & 0 \\ 0 & 0 & -\dfrac{h_3 \rho_{vbr}}{\varepsilon_{SMA}} \end{bmatrix} \tag{98}$$

Choosing above parameters according to [10], we have:

$$\begin{cases} h_2 = h_3 = 0.5 \\ \rho_{vbl} = \rho_{vbr} = 0.08 \\ \varepsilon_{SMA} = 0.1 \end{cases} \tag{99}$$

And other parameters are selected like the 4th order case, then:

$$A = \begin{bmatrix} 0 & 31.4159 & 0 & 0 & 0 & 0 \\ -31.4159 & -0.1000 & 0 & 2.0000 & 0 & 0 \\ 0 & 0 & -0.1000 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & -0.4000 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.0000 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 32.7761 & 0 & 0 \\ 0 & 0.4000 & 0 \\ 0 & 0 & -0.4000 \end{bmatrix} \tag{100}$$

eigenvalues of linerized system are:

$$\begin{cases} -0.0500 + 31.4159i \\ -0.0500 - 31.4159i \\ -0.1000 \\ -0.4000 \\ -1.0000 \\ -1.0000 \end{cases} \tag{101}$$

All the eigenvalues have negative real part and then, linearized system is stable.

Step3.

$$Q = R = I \tag{102}$$

Where, I is identity matrix.

Step4. By the means of LQR:

$$K = \begin{bmatrix} 0.8700 & -1.0250 & -0.9019 & -1.0742 & 0.0000 & -0.0000 \\ 0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.1926 & -0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.1926 \end{bmatrix} \tag{103}$$

Step5.

$$Max(\lambda_{A_K}) = -1.0034 \Rightarrow$$
$$c = 0.9030 \tag{104}$$

Step6. Solving Lyapanov Equation for selected parameters above yields:

$$P = \begin{bmatrix} 8.0787 & 0.0540 & 0.5265 & -0.2271 & 0.0000 & -0.0000 \\ 0.0540 & 8.5553 & 0.4820 & 0.2856 & 0.0000 & -0.0000 \\ 0.5265 & 0.4820 & 9.6056 & 0.2916 & 0.0000 & -0.0000 \\ -0.2271 & 0.2856 & 0.2916 & 0.0559 & 0.0000 & -0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 2.9802 & -0.0000 \\ -0.0000 & -0.0000 & -0.0000 & -0.0000 & -0.0000 & 2.9802 \end{bmatrix} \tag{105}$$

Step7. For new local controller to satisfy the input bounds (55) and (99), since U is large enough this can not reduce the terminal region; in fact, in the above region the local feedback controller with gain of step 3 is admissible; hence, with this new region of attraction:

$$r_1 = 41.1410 \tag{106}$$

Step8. In this step it is required to obtain the disturbance parameter (μ). In the delta wing model uncertainty consists of term $C_l(x_{vb})$; simulation results show that

$$Max(C_l(x_{vb})) = 0.01 \tag{107}$$

That barely happens and typically is about 0.001 for region of attraction (55) and (99), we have:

$$\|x\| \le 6.28 \tag{108}$$

Then:

$$\mu = \frac{Max(C_l)}{\|x\|} = 0.0016 \tag{109}$$

However for some other perturbations which are negligible in comparison with this unmodeled uncertainty and to avoid selecting a conservative bound on the uncertainties, we choose:

$$\mu = 0.002 \tag{110}$$

Considering the delay term as the nonlinearity, μ_ϕ is computed as following:

$$\mu_\phi = Max(\frac{\|x_4\|}{\|x\|}) = 0.2771 \tag{111}$$

Since in our case the nonlinearity is weak, then no considerable changes happen in r:

$$r \cong r_1 = 41 \tag{112}$$

Step9. Again like 4th order we want a smaller final horizon time; then, $T=1$; and also $L=6$ and:

$$l = Max\|x\|_{\infty} = 1.74 \tag{113}$$

Hence:

$$r_{new} = 26.82 \tag{114}$$

Step10. Figure 11 shows the trajectory of nominal system with the design parameters above and open loop optimal control; the final trajectory of system reaches:

$$x(t+T;\, x(t),\, t) = [-0.0126 \quad 0.0010 \quad 0.6023 \quad -0.2194 \quad 0.2926 \quad -0.2926] \tag{115}$$

Then:

$$x^T(t+T)Px(t+T) = 3.9133 < r_{new} \tag{116}$$

Consequently, for MIMO system, like 4th order, there is a feasible solution and with the designed parameters the closed loop MIMO vortex break down roll dynamics of delta wing is stabilizable.

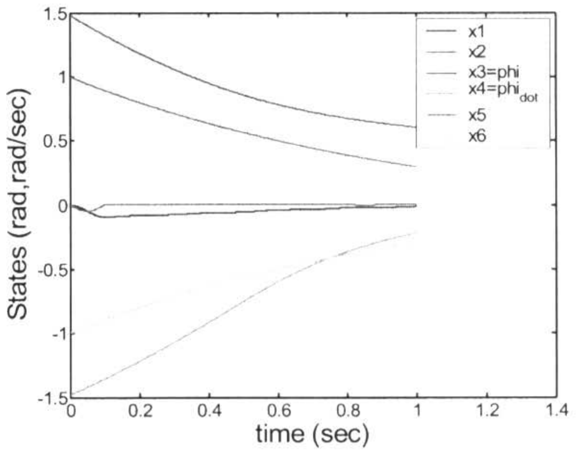


Fig11. Open loop trajectory

3.9 Appendix E: Delta Wing Dynamics Coupled with New SMA Antagonistic Actuation System

The delta wing vortex coupled model will be as follows:

$$\begin{cases} \dot{x}_1(t) = c\,x_2(t) \\ \dot{x}_2(t) = -c\,x_1(t) - \varepsilon_d\,x_2(t) + x_4(t) + x_4(t-T) \\ \dot{x}_3(t) = -\varepsilon_d\,x_3(t) + x_4(t) \\ \dot{x}_4(t) = -C_l(\aleph)Q - f_c\,x_4(t) + h_1 u_1(t)/I_w \end{cases} \tag{117}$$

Where

$$\aleph = [X_{vbl}(t), X_{vbr}(t)] \tag{118}$$

$$X_{vbl}(t) = X_{sl}(x(t)) + X_{sl}(x(t))k_q(t)x_4(t) + a(t)x_1(t) + y_{smal}(t) \tag{119}$$

$$X_{vbr}(t) = X_{sr}(x(t)) + X_{sr}(x(t))k_q(t)x_4(t) - a(t)x_1(t) + y_{smar}(t) \tag{120}$$

The discrete-time dynamics of new SMA model developed in [15] and it is converted to continuous-time in [16]. The conceptual flowchart of the delta wing vortex-coupled dynamics with the new SMA model is shown in figure 12.

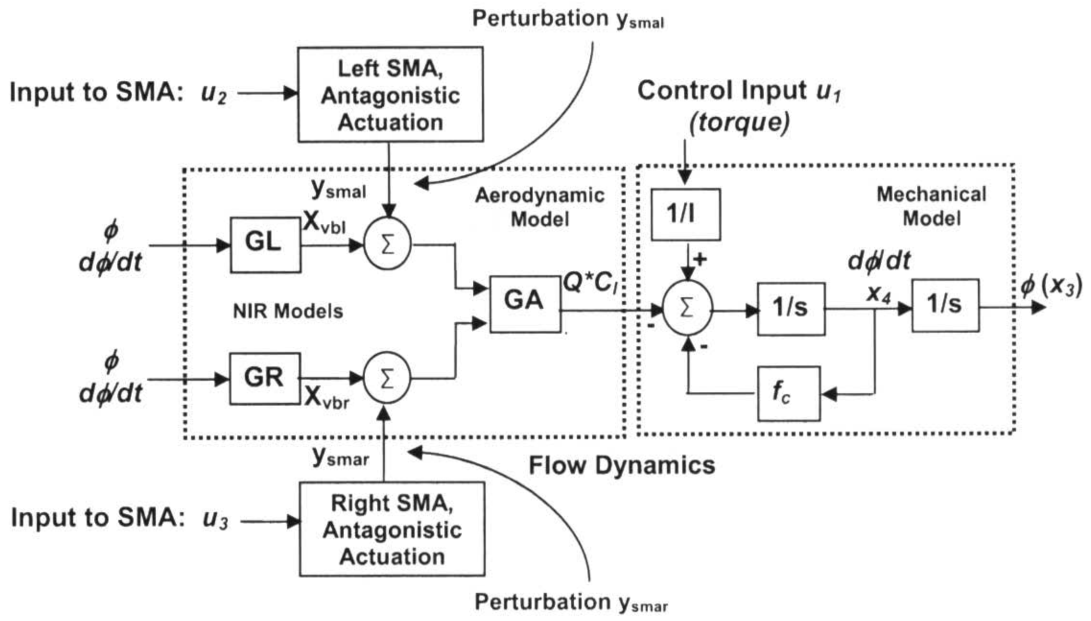


Fig. 12 Conceptual flowchart of the delta wing vortex-coupled dynamics with the new SMA model

The Simulink block diagram of the *continuous time* model of SMA control loop is shown in figure 13. The SMA control loop has three main blocks: 1) low pass filter; 2) control law with an integrator; 3) SMA model (from identification process).

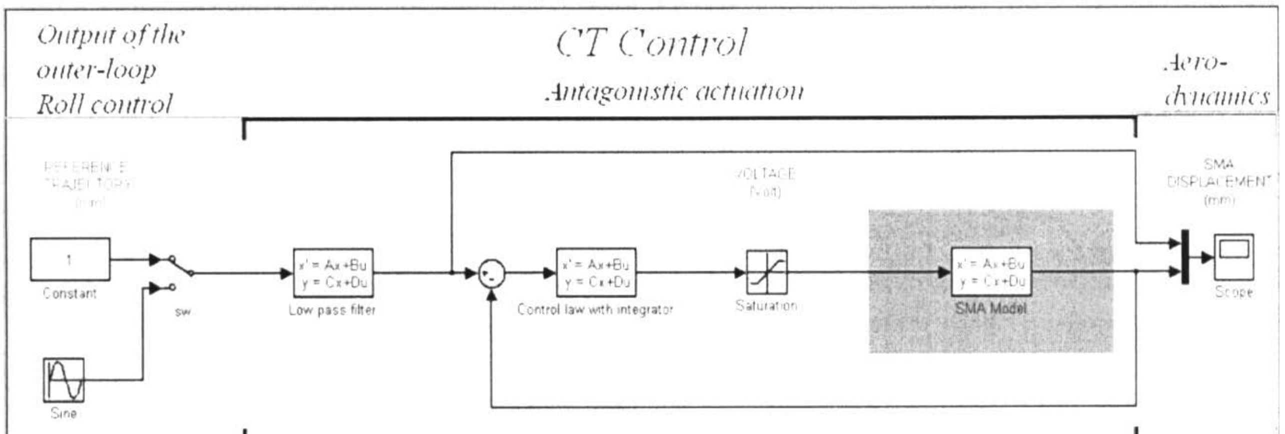


Fig.13. Simulink block diagram of the *continuous time* model of SMA control loop

The SMA control loop contains the continuous time model of the following items:

1- Low pass filter:

$$\begin{aligned} \dot{x}_{lpf}(t) &= A_{lpf}x_{lpf}(t) + B_{lpf}u_i(t) \\ y_{lpf}(t) &= C_{lpf}x_{lpf}(t) + D_{lpf}u_i(t) \end{aligned} \tag{121}$$

Where

$$u_i(t) = u_{2-0r-3}(t) \tag{122}$$

The numerical values of the matrices for the continuous time model are as follows:

$$A_{lpf} = \begin{bmatrix} -20.0000 & -100.0000 \\ 01.0000 & -000.0000 \end{bmatrix},\ B_{lpf} = \begin{bmatrix} 1.0000 \\ 0.0000 \end{bmatrix},\ C_{lpf} = [0.0000 \quad 100.0000],\ D_{lpf} = [0.0000] \tag{123}$$

2- Control law with an integrator:

$$\begin{aligned} \dot{x}_{cont}(t) &= A_{cont}x_{cont}(t) + B_{cont}u_{cont}(t) \\ y_{cont}(t) &= C_{cont}x_{cont}(t) + D_{cont}u_{cont}(t) \end{aligned} \tag{124}$$

Where

$$u_{cont}(t) = (y_{lpf}(t) - y_{sma}(t)) \tag{125}$$

The numerical values of the matrices for the continuous time model are as follows:

$$A_{cont} = \begin{bmatrix} 0.0000 & 108.5991 & -117.8455 \\ -0.0000 & 174.8300 & 226.4447 \\ 0.0000 & -177.2327 & -223.8374 \end{bmatrix},\ B_{cont} = \begin{bmatrix} 27.6270 \\ 1.9049 \\ -2.4398 \end{bmatrix},\ C_{cont} = [1.0000 \ 1.0000 \ 0.0000],\ D_{cont} = [0.1433] \tag{126}$$

3- SMA model from system identification process:

$$\begin{aligned} \dot{x}_{sma}(t) &= A_{sma}x_{sma}(t) + B_{sma}u_{sma}(t) \\ y_{sma}(t) &= C_{sma}x_{sma}(t) + D_{sma}u_{sma}(t) \end{aligned} \tag{127}$$

Where

$$u_{sma}(t) = sat(y_{cont}(t)) \tag{128}$$

The saturation function has been defines as follows:

$$sat(y_{cont}(t)) = \begin{cases} y_{cont}(t) \geq 6 \rightarrow y_{cont}(t) = 6 \\ -6 \leq y_{cont}(t) \leq 6 \rightarrow y_{cont}(t) = y_{cont}(t) \\ y_{cont}(t) \leq -6 \rightarrow y_{cont}(t) = -6 \end{cases} \tag{129}$$

The numerical values of the matrices for the continuous time model are as follows:

$$A_{sma} = \begin{bmatrix} 177.3034 & -177.4760 \\ 224.4827 & -224.2960 \end{bmatrix},\ B_{sma} = \begin{bmatrix} 107.8497 \\ -116.6330 \end{bmatrix},\ C_{sma} = \begin{bmatrix} -0.0195 & 0.0210 \end{bmatrix},\ D_{sma} = \begin{bmatrix} 0.0000 \end{bmatrix} \tag{130}$$

A robust quasi RHC controller is designed for the delta wing dynamics with new SMA model; the output of this RHC controller is u_1; and u_{sma} is set to be equal to 10% of u_1. All design parameters are the same as 4th order case and the time delay for simulations is $\tau = 0.1 Sec$. Simulation results are depicted in figures 14 through 17; these figures show that with the new SMA model the robust quasi RHC can successfully control the vortex coupled dynamics of delta wing aircraft.

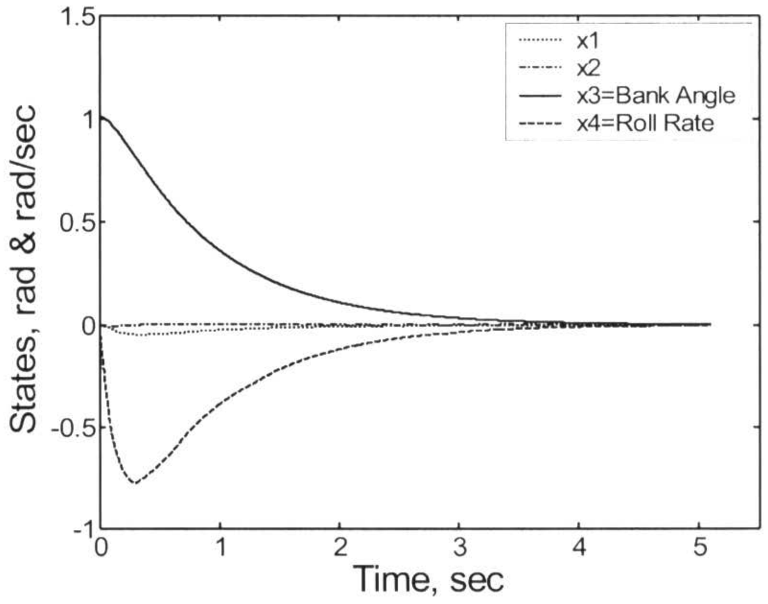


Fig14. Time history of states with new SMA model

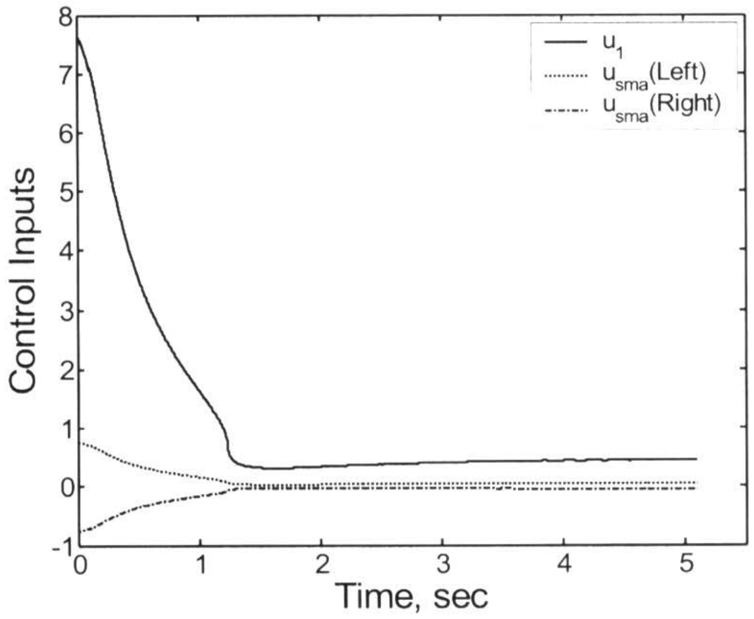


Fig15. Time history of control inputs with new SMA model

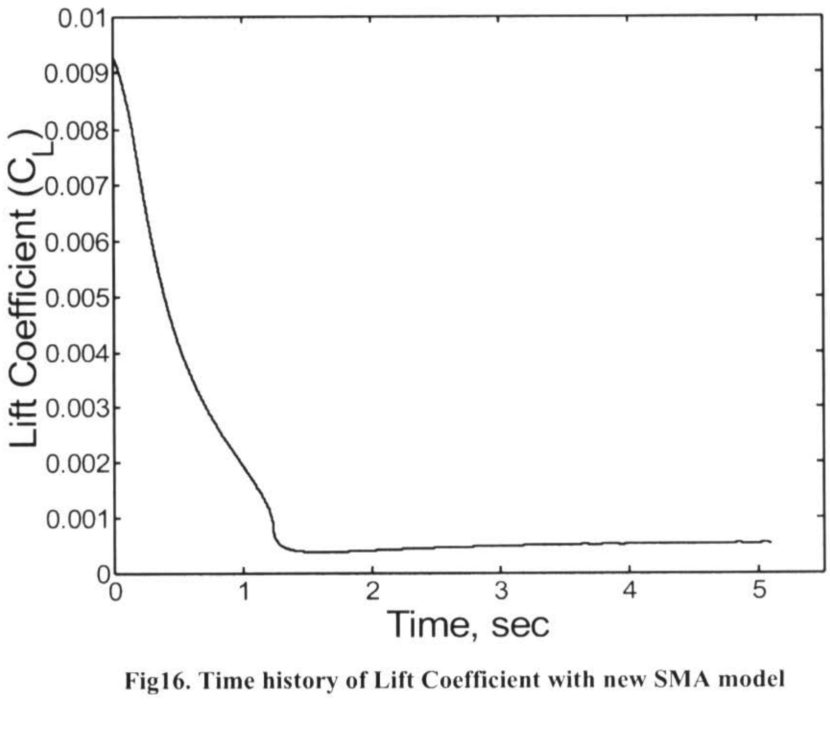


Fig16. Time history of Lift Coefficient with new SMA model

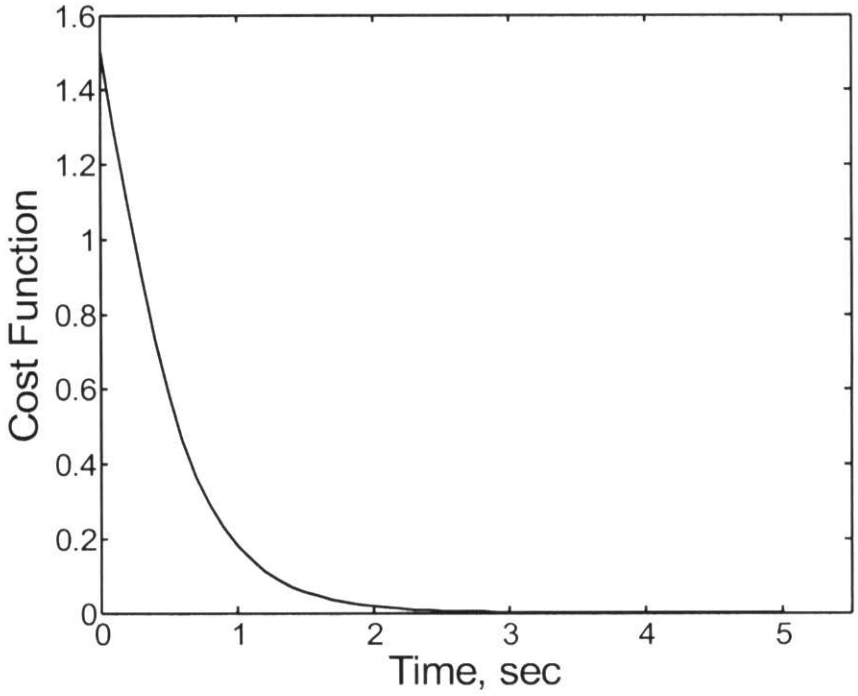


Fig17. Time history of cost function with new SMA model

3.10 References

[1] William Dunbar, "Distributed Receding Horizon Control of Multi agent Systems", PhD thesis, Control and Dynamical Systems, California Institute of Technology, April, 2004.

[2] M. B. Milam, R. Franz, J. E. Hauser, R. M. Murray, "Receding horizon control of a vectored thrust flight experiment", submitted to IEE Proceedings on Control Theory and Applications.

[3] H.Chen, F.Allgower, "A Quasi Infinitive Horizon Nonlinear Model Predictive Control Scheme with Guaranteed Stability", Automatica, Vol. 34, No. 10, pp 1205-1217,1998.

[4] H. Michalska, D.Q. Mayne, ' Robust Receding Horizon Control of Constrained Nonlinear Systems', IEEE Transactions on Automatic Control, Vol. 38, No.11, November 1993, PP. 1623-1633.

[5] L. Magni, H. Nijmeijer, A. J. van der Schaft, "A Receding Horizon approach to the nonlinear H_∞", Automatica, Vol 37, 2001, PP.429-435.

[6] M. B. Milam, R. Franz, J. E. Hauser, R. M. Murray, "Receding horizon control of a vectored thrust flight experiment", submitted to IEE Proceedings on Control Theory and Applications.

[7] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems", PhD Thesis, California Institute of Technology, Pasadena, CA, 2003.

[8] A. Jadbabaie, J. Yu and J. Hauser, "Unconstrained Receding-Horizon Control of Nonlinear Systems", IEEE Transactions on Automatic Control, Vol.46, No. 5, May 2001, PP. 776-783.

[9] J. J. Slotine, W. Li, Applied Nonlinear Control, Prentice Hall, 1991.

[10] Pakmehr M., Gordon B.W. and Rabbath C.A.,"Control Oriented Modeling and Identification of Delta Wing Vortex-Coupled Roll Dynamics", to be presented at the American Control Conference 2005, Portland, Oregon, June 8-10, 2005 (to be published)

[11] M. Pakmehr, B.W.Gordon, "Innovative Methods of Flight Control by Manipulation of the Flow Structure" Internal report No.5, submitted to DRDC.

[12] X. Z. Huang, "Non-Linear Indicial Response and Internal State-Space (NIRISS) Representation and Its Application on Delta Wing Configurations", RTO Technical Report, RTO-TR-047.

[13] Philip E. Hill, Walter Murray, and Margaret H. Wright, Practical optimization, Academic Press, page 65-66, 1981.

M. J. D. Powell , "Introduction to Constrained Optimization", *Numerical Methods for Constrained Optimization*, Ed. P. E. Gill, W. Murray, Academic Press, 1974.

[14] H. Izadi, M. Pakmehr, B. W. Gordon, C. A. Rabbath, "A Receding Horizon Control Approach for Roll Angle Tracking of Delta Wing Vortex-Coupled Dynamics", Proceedings of IEEE Aerospace conference, March 3-10, 2007, Big Sky, MT, USA.

[15] Léchevin, N., Boissoneault, O., and Rabbath, C. A., "Control of Antagonistic Actuator by Means of Shape Memory Alloy - Identification-Based Control", Internal Report, Defence Research and Development Canada (DRDC), Valcartier, Quebec, Canada, January 5, 2006.

[16] Pakmehr M., Gordon B. W., "Hybrid Control Scheme and LPV Sliding Mode Control for Delta Wing Roll Dynamics Coupled with SMA Micro Actuator Dynamics", internal report, submitted to DRDC, March 2006.

CHAPTER 4: FULLY DECENTRALIZED RECEDING HORIZON CONTROL (RHC) ALGORITHMS FOR COOPERATIVE PATH PLANNING AND TRAJECTORY GENERATION OF MULTI-VEHICLE SYSTEMS

(Task # 6)

Hojjat A. Izadi
PhD Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory

March 2006

Abstract

This report documents the work related to task #6 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". The decentralized control problem of multi-UAVs (Unmanned Aerial Vehicle) consists of three main problems: inner-loop controller design, task assignment and path planning. In this report, optimization based approaches have been used to study these problems. First, a receding horizon controller (RHC) is designed to stabilize the inner loop of an under-actuated 3DOF helicopter dynamics. Then, another receding horizon optimization based approach will be used for path planning of a fleet of UAVs. An optimization based task assignment is also designed which minimizes an overall cost to go. The uniqueness of the decisions made by all UAVs is guaranteed by aiming the fact that the global minimum of the cost function is unique for all UAVs. In this report, it is assumed that the path planner is fully decentralized and there is no communication during performing the mission. Also, it is assumed that all UAVs know the initial position of team members and targets a priori. This architecture will be triggered whenever the communication is not available or the communicated information is not reliable.

Nomenclature

h_{mr}	Main rotor hub height above c.g	P, Q, R	Weight matrices of quadratic cost fur
h_{tr}	Tail rotor height above c.g	P, q, r	Angular velocities
I_{xx}	rolling moment of inertia	S_x^{fus}	frontal fuselage drag area
I_{yy}	pitching moment of inertia	S_y^{fus}	side fuselage drag area
I_{zz}	yawing moment of inertia	S_z^{fus}	vertical fuselage drag area
K_β	hub torsional stiffness	T	finite horizon time
l_{ht}	stabilizer location behind c.g	u, v, w	Local velocities
l_{tr}	Tail rotor hub location behind c.g	β	Side slip angle
m	helicopter mass	ψ, ϕ, θ	Euler angles
N	Number of UAVs		

Subscripts

b	Body axis (coordinate)	mr	Main rotor
c	Computation	tr	Tail rotor
s	Stability axis (coordinate)	vf	Vertical fin
fus	Fuselage	r	Real time
ht	Horizontal tail		

4.1 Introduction

This report addresses the following task related to the project "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques":

Task 6: To develop time-constrained decentralized receding-horizon control (RHC) algorithms for cooperative path planning and trajectory generation for multi-vehicle systems. Develop the cost functions, select the numerical methods for optimization solution, and apply to aerial vehicle models as available from the open literature.

UAVs offer numerous advantages to the military and civil applications. Most notable in the military are the advantages of the ability to perform missions classified as "dull, dirty, or dangerous" [1]. Missions that are classified as dull include examples of an aircraft loitering over airspace for long periods of time while providing surveillance or jamming enemy electronic devices. These types of missions can last for especially long periods of time, such that manned crews would not be optimal to perform. The second type of mission is the dirty type; this type of mission includes reconnaissance in areas that have been contaminated by nuclear, biological, or chemical weapons, where the presence of manned aircraft would put the crew in danger. The last type is the dangerous mission, such as high-risk but high-value targets or Suppression of Enemy Air Defences (SEAD). These wide applications of multi-UAVs missions motivate researchers to work on the cooperative control of multi-UAVs; however, this cooperation must be decentralized as much as possible to increase the autonomy of each UAV for performing the mission [2]. Reduction of communication and making the decision autonomously are two main results of the decentralized control.

Figure (1) shows a typical cooperative control architecture for each UAV. Depending on the path planning method or control method this architecture may change; for instance in some RHC based architectures the control action design and path planning are done simultaneously as a result of minimization of a cost function. The cooperative control of multi-UAVs consists of three main parts [3] as it is shown in figure 1: inner loop controller design, task assignment unit and path planning unit:

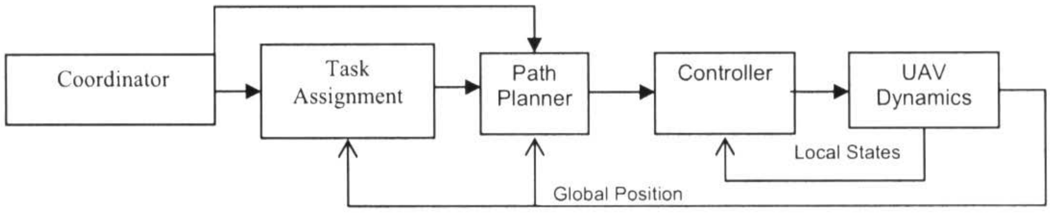


Fig 1. Block diagram of individual UAV in cooperative control problem

The arrows show the flow of information in the above diagram. In this diagram, the coordinator behaves like an autopilot and based on the information it receives from the UAV dynamics and mission manager (which can be a ground station), sends some commands to the task assignment and path planner units. Based on the information about the current position of UAV and the position of the targets, the Task Assignment unit decides which task should be performed first and inputs the position of that task to Path Planner unit. Afterward, the Path Planner Unit plans the paths for each UAV with considering the collision avoidance with neighbour UAVs. Based on the path planned in the Path Planner Unit, the inner loop controller inputs the control action to the UAV dynamics so that the planned path is followed by UAV. This procedure is performed on-line by each UAV during performing the mission.

Since designing an inner loop controller requires a model of UAV, section 4.2. deals with the modeling of a small-scale rotorcraft UAV and deriving the equations of motion. The dynamics of a helicopter is a nonlinear and under-actuated dynamics which is subject to the saturations on the control inputs and states. To control this complex dynamics, an optimization based method known as receding horizon control (RHC) or model predictive control (MPC) will be used to design a tracking controller. The RHC can easily handle the constraints on both inputs and states and it is a promising method for designing feedback controller for the nonlinear systems [4]. However, the main drawback of the RHC is the high computational burden that makes it hard to be applied to the on-line fast applications. Hence, the advantages of the flat outputs and B-Spline functions are used to alleviate the computational burden during the optimization process. The idea of flat outputs is discussed in Appendix A.

Although the task assignment unit is not in the scope of this report, it is discussed shortly; because, the task assignment unit is an important element of the cooperative control and it is needed for understanding the cooperative scenarios. Furthermore, the task assignment and path planning are strongly coupled, and optimal coordination plans can not be achieved if this coupling is ignored [3]. The task assignment unit will be addressed in section 2.2. Based on the available information on the initial position of the UAVs, targets and obstacles, the task assignment unit assigns one (or more) target to each UAV. It is desired to develop a task assignment unit which is able to allocate tasks under imperfection communication and is reliable in presence of uncertain information because in the real world there is often a degree of uncertainty in the information and the perfect information and communication is not available; hence, the task assignment unit should be able to perform the mission in presence of uncertainties. In the task assignment unit designed in this report, an overall cost to go for all UAVs will be minimized to allocate the targets, and UAVs need only the initial position of all UAVs and targets.

As soon as the tasks are allocated to the UAVs, the path planning begins; in fact, the task assignment unit decides about the destination of the UAVs and informs the UAVs about the destination of all UAVs. Based on the decision made by the task assignment unit, the path planning unit designs the paths for all UAVs; the paths starts from the initial position of the UAVs and have to visit the targets while avoiding the obstacles. Again like the task assignment unit, in the path planning unit it is tried to reduce the amount of information communicated among the UAVs. The planned paths are fed to the controller (figure 1) to provide the control signal such that the UAVs follow the planned paths. In this report, a receding horizon optimization method is used for path planning.

This report consists of three main chapters, in the first chapter, the small-scale UAV rotorcraft dynamics will be presented and for more efficiency in the tracking problems it will be presented in the stability coordinate [5]. Then, by the means of B-Spline basis functions, an RHC is designed to control the inner loop 3DOF under-actuated dynamics of helicopter. In the second chapter, after reviewing some common scenarios in the recent control literatures a typical scenario for cooperative control will be proposed. An optimization based task assignment unit is also designed and finally different methods in the path planning of UAVs are reviewed. In the third chapter, a decentralized receding horizon optimization based path planner is proposed to perform the proposed scenario. In all cases the simulation results show that the decentralized receding horizon path planner can generate the paths for multi-UAVs appropriately and the path planner fulfills all requirements.

4.2 Inner-loop control design

In this section, an optimization-based receding horizon controller is designed to control the local dynamics of the helicopter. RHC because of its flexibility and prominent ability to handle the constraints is a good candidate to control the aerospace vehicles which are nonlinear, constrained, and uncertain and may exhibit under-actuated dynamics... The optimality of the RHC is also a considerable advantage.

Firstly, the 6DOF helicopter dynamics, forces and moments in body coordinate is introduced from the open literature; then, to ease the outer loop tracking control, the dynamics is transferred to the stability coordinate. After that, a formulation of quasi-infinite-RHC is presented and by the means of B-Spline basis functions and advantages of flat outputs, the RHC is implemented to the nonlinear 3DOF dynamics of helicopter for both stabilization and tracking cases.

4.2.1 Review of Helicopter Modelling Literature

A mathematical model of the helicopter is required to design and simulate the control system. A sophisticated work was done by MIT [6], where they developed an analytical and low-order dynamic model of a miniature aerobatic helicopter in body coordinate. The model includes states for the longitudinal and lateral states including positions, velocities, angular velocities and also the control inputs including main rotor and tail rotor forces. Adding the Euler angles to the dynamic equations makes it possible to make a relation between the local and global coordinate parameters for position tracking problems. The dynamic forces and moments which act on the helicopter are shown in figure 2. Using the Newton's second law for 6 degree of freedom (6DOF) yields the following dynamic equation in body coordinates [6]:

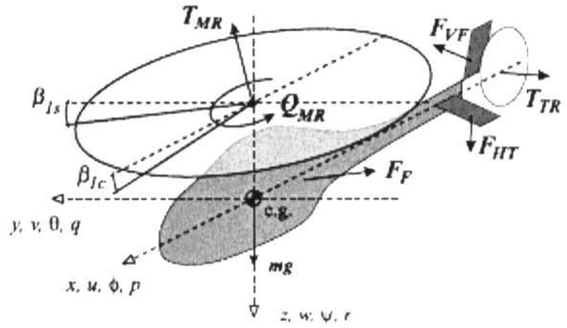


Fig2. Helicopter forces and moments

$$\begin{cases} \dot{u} = vr - wq - g\,Sin\theta + (X_{mr} + X_{fus})/m \\ \dot{v} = wp - ur + gSin\,\theta\,Cos\,\phi + (Y_{mr} + Y_{fus} + Y_{tr} + Y_{vf})/n \\ \dot{w} = uq - vp + gCos\,\phi\,Cos\,\psi + (Z_{mr} + Z_{fus} + Z_{ht})/m \\ \dot{p} = qr(I_{yy} - I_{zz})/I_{xx} + (L_{mr} + L_{vf} + L_{tr})\,/I_{xx} \\ \dot{q} = pr(I_{zz} - I_{xx})/I_{yy} + (M_{mr} + M_{ht})\,/I_{yy} \\ \dot{r} = pq(I_{xx} - I_{yy})/I_{zz} + (-Q_e + N_{vf} + N_{tr})\,/I_{zz} \end{cases} \qquad (1)$$

Where, the set of forces and moments acting on the helicopter are organized by components: $()_{mr}$ for the main rotor, $()_{tr}$ for the tail rotor, $()_{fus}$ for the fuselage (includes fuselage aerodynamic effects), $()_{vf}$ for the vertical fin and $()_{ht}$ for the horizontal stabilizer; also, Q_e is the engine torque (positive clockwise). These forces and moments are discussed separately below.

4.2.1.1 Main rotor forces and moments

The following forces and moments act on the helicopter due to the main rotor thrust:

$$\begin{cases} X_{mr} = K_{mr_x} T_{mr} \\ Y_{mr} = b_1 T_{mr} \\ Z_{mr} = T_{mr} \\ L_{mr} = (K_\beta + T_{mr} h_{mr})\, b_1 \\ M_{mr} = (K_\beta + T_{mr} h_{mr})\, a_1 \end{cases} \qquad (2)$$

Where, K_β is the hub torsional stiffness, h_{mr} is the main rotor hub height above c.g., a_1 and b_1 are coefficients related to the main rotor flapping angle and the blade azimuth angle; the typical value of these parameters for MIT mini helicopter are shown in table 1.

4.2.1.2 Fuselage forces and moments

The fuselage forces can be approximated by:

$$\begin{cases} V_\infty = \sqrt{u_a^2 + v_a^2 + (w_a + v_{imr})^2} \\ V_{imr} = \sqrt{\dfrac{mg}{2\rho\pi R_{mr}^2}} \\ X_{fus} = -0.5\rho S_x^{fus} u_a v_\infty \\ Y_{fus} = -0.5\rho S_y^{fus} v_a v_\infty \\ Z_{fus} = -0.5\rho S_z^{fus} (w_a + v_{imr}) v_\infty \end{cases} \tag{3}$$

Where, V_∞ is the overall velocity of helicopter, ρ is the air density, S_x^{fus}, S_y^{fus} and S_z^{fus} are effective frontal, side and vertical drag areas of the fuselage, u_a, v_a and w_a are fuselage center of pressure velocities with respect to air (i.e. $u_a = u - u_w$, where u_w is the projection of wind velocity vector on the X body axis).

4.2.1.3 Tail Rotor and Vertical Fin forces and moments

The contribution of tail rotor and vertical fin is the following forces and moments:

$$\begin{cases} Y_{Tr} = T_{tr} \\ N_{tr} = -T_{tr} l_{tr} \\ L_{tr} = T_{tr} h_{tr} \\ N_{vf} = -Y_{vf}\, l_{tr} \\ L_{vf} = Y_{vf} h_{tr} \end{cases} \tag{4}$$

Where, l_{tr} is the tail rotor hub location behind c.g. and h_{tr} is the tail rotor height above c.g. and Y_{vf} is the aerodynamic force generated by vertical fin. Table 1 shows the physical MIT mini helicopter parameters [6]:

Table 1: Parameters of MIT Instrumented X-Cell 60 SE Helicopter

Parameter	**Description**
$m = 8.2$ kg	helicopter mass
$I_{xx} = 0.18$ kg m^2	rolling moment of inertia
$I_{yy} = 0.34$ kg m^2	pitching moment of inertia
$I_{zz} = 0.28$ kg m^2	yawing moment of inertia
$K_\beta = 54$ N·m/rad	hub torsional stiffness
$\gamma_{fb} = 0.8$	stabilizer bar Lock number
$B_{\delta_{lat}}^{nom} = 4.2$ rad/rad	lateral cyclic to flap gain at nominal rpm
$A_{\delta_{lon}}^{nom} = 4.2$ rad/rad	longitudinal cyclic to flap gain at nominal rpm
$K_\mu = 0.2$	scaling of flap response to speed variation
$\Omega_{nom} = 167$ rad/sec	nominal m.r. speed
$R_{mr} = 0.775$ m	m.r. radius
$c_{mr} = 0.058$ m	m.r. chord
$a_{mr} = 5.5$ rad^{-1}	m.r. blade lift curve slope
$C_{D_0}^{mr} = 0.024$	m.r. blade zero lift drag coefficient
$C_{T_{max}}^{mr} = 0.0055$	m.r. max thrust coefficient
$I_{\beta_{mr}} = 0.038$ kg m^2	m.r. blade flapping inertia
$R_{tr} = 0.13$ m	t.r. radius
$c_{tr} = 0.029$ m	t.r. chord
$a_{tr} = 5.0$ rad^{-1}	t.r. blade lift curve slope
$C_{D_0}^{tr} = 0.024$	t.r. blade zero lift drag coefficient
$C_{T_{max}}^{tr} = 0.05$	t.r. max thrust coefficient
$n_{tr} = 4.66$	gear ratio of t.r. to m. r.
$n_{es} = 9.0$	gear ratio of engine shaft to m. r.
$\delta_r^{trim} = 0.1$ rad	t.r. pitch trim offset
$S_{vf} = 0.012$ m^2	effective vertical fin area
$C_{L_\alpha}^{vf} = 2.0$ rad^{-1}	vertical fin lift curve slope
$\epsilon_{vf}^{tr} = 0.2$	fraction of vertical fin area exposed to t.r. induced velocity
$S_{ht} = 0.01$ m^2	horizontal fin area
$C_{L_\alpha}^{ht} = 3.0$ rad^{-1}	horizontal tail lift curve slope
$P_{eng}^{idle} = 0.0$ Watts	engine idle power
$P_{eng}^{max} = 2000.0$ Watts	engine max power
$K_p = 0.01$ sec/rad	proportional governor gain
$K_i = 0.02$ 1/rad	integral governor gain
$f_p^s = 12.5$ Hz	rolling resonance frequency of the suspension system
$f_q^s = 9.0$ Hz	pitching resonance frequency of the suspension system
$f_r^s = 9.6$ Hz	yawing resonance frequency of the suspension system
$\zeta^s = 0.05$	damping ratio of the suspension system material
$S_x^{fus} = 0.1$ m^2	frontal fuselage drag area
$S_y^{fus} = 0.22$ m^2	side fuselage drag area
$S_z^{fus} = 0.15$ m^2	vertical fuselage drag area
$h_{mr} = 0.235$ m	m.r. hub height above c.g.
$l_{tr} = 0.91$ m	t.r. hub location behind c.g.
$h_{tr} = 0.08$ m	t.r. height above c.g.
$l_{ht} = 0.71$ m	stabilizer location behind c.g.

4.2.2 3DOF UAV dynamics in body coordinate

In order to simplify the problem for designing the controller, the helicopter dynamics will be reduced to 3DOF dynamics. Hence, the helicopter won't have movement in the vertical direction and roll and pitch orientations; this will make the following parameters to zero:

$$\begin{cases} w=0 \\ \phi=0 \\ p=0 \\ \theta=0 \\ q=0 \end{cases} \tag{5}$$

Setting the above parameters related to the vertical position, rolling and pitching orientations to zero in (1) yields:

$$\begin{cases} \dot{u}=vr+c_x u\sqrt{u^2+v^2}-\dfrac{k_{mr_x}}{m}T_{mr} \\ \dot{v}=-ur+c_y v\sqrt{u^2+v^2} \\ \dot{r}=\dfrac{-l_{tr}}{I_{zz}}T_{tr} \\ \dot{\psi}=r \end{cases} \tag{6}$$

Where:

$$\begin{cases} c_x=-\dfrac{1}{m}(0.5)\rho_a S_{fus_x} \\ c_y=-\dfrac{1}{m}(0.5)\rho_a S_{fus_y} \end{cases} \tag{7}$$

The last equation in (6) comes from the Euler angle equations. For the positions in the global coordinate we have:

$$\begin{cases} \dot{x}_c=u\cos(\psi)-v\sin(\psi) \\ \dot{y}_c=u\sin(\psi)+v\cos(\psi) \end{cases} \tag{8}$$

The main drawback of this dynamics is that for tracking problems as we feed the desired trajectory to the system, from (8), we have 2 equations and 3 unknowns; therefore, we write the dynamics in a coordinate called stability coordinate.

4.2.3 Derivation of Flight Equations in Stability Coordinate

In the stability coordinate, the x-axis is set to be in the direction of the velocity of the helicopter [5]. In figure 3 the relative location and orientation of the stability coordinate and body coordinate are shown for 3DOF dynamics.

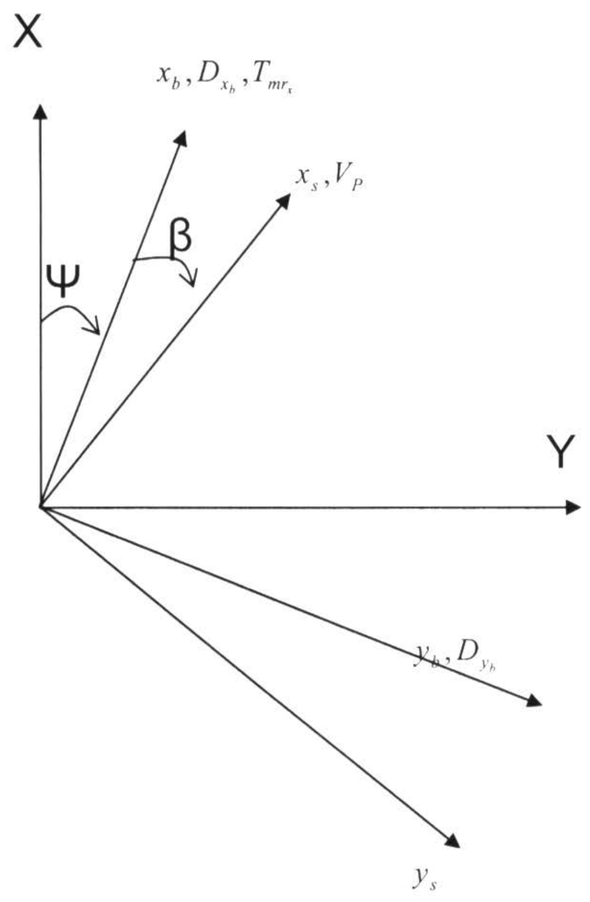


Fig. 3. Body and stability coordinate

To obtain the helicopter dynamics in the stability coordinate, we write the Newton's second law in stability coordinate:

$$\sum F_{x_s} = ma_{x_s} \Rightarrow T_{mr_x} Cos\beta + D_{x_b} Cos\beta + D_{y_b} .Sin\beta = m\ddot{x}_s \qquad (9)$$

$$\sum F_{y_s} = ma_{y_s} \Rightarrow -T_{mrx} Sin\beta - D_{x_b} Sin\beta + D_{y_b} .Cos\beta = m\ddot{y}_s \qquad (10)$$

Where, D is the drag force. From the MIT report [6] for the drag forces we have:

$$D_{x_b} = (-0.5)\times S_{fus_x}\rho u_b\sqrt{u_b^2 + v_b^2} \tag{11}$$

$$D_{y_b} = (-0.5)\times S_{fus_y}\rho v_b\sqrt{u_b^2 + v_b^2} \tag{12}$$

The relation between the velocities in the stability coordinate and body coordinate is:

$$\begin{aligned} u_b &= u_s Cos\beta \\ v_b &= u_s Sin\beta \end{aligned} \tag{13}$$

Using (13) in (11) and (12), we will have:

$$D_{x_b} = C_x m u_s^2 Cos\beta \tag{14}$$

$$D_{y_b} = C_y m u_s^2\, Sin\beta \tag{15}$$

Where:

$$\begin{aligned} C_x &= (-0.5) * S_{fus_x}\rho / m \\ C_y &= (-0.5) * S_{fus_y}\rho / m \end{aligned} \tag{16}$$

Also, from the MIT report [6] for the main rotor force along the x-axis we have:

$$T_{mr_x} = -k_{mr_x} T_{mr} \tag{17}$$

Then, Putting Equations (14), (15) and (17) in (9) and (10) yields:

$$-k_{mr_x} T_{mr} Cos\beta + C_x m u_s^2 Cos^2\beta + C_y m u_s^2 Sin^2\beta = m\ddot{x}_s \tag{18}$$

$$k_{mr_x} T_{mr} Sin\beta - C_x m u_s^2 Cos\beta\, Sin\beta + C_y m u_s^2 Sin\beta.Cos\beta = m\ddot{y}_s \tag{19}$$

Now, we must find the components of the acceleration in x_s and y_s directions; since in the stability coordinate the component of velocity in y_s direction is zero, we have:

$$\vec{V} = u_s\vec{i}_s + 0\vec{j} \tag{20}$$

Where $\vec{i}_s$ and $\vec{j}_s$ are the unit vectors. Differentiation of (20) yields:

$$\vec{a} = \frac{d\vec{V}}{dt} = \dot{u}_s\vec{i}_s + u_s\dot{\vec{i}}_s = \dot{u}_s\vec{i}_s + u_s(\dot{\psi} + \dot{\beta})\vec{j}_s \tag{21}$$

Putting the acceleration components into (18) and (19) from (21), we will have:

$$-k_{mr_x} T_{mr} Cos\beta + C_x m u_s^2 Cos^2\beta + C_y m u_s^2 Sin^2\beta = m\dot{u}_s \tag{22}$$

$$k_{mr_x} T_{mr} Sin\beta - C_x mu_s^2 Cos\beta\, Sin\beta + C_y mu_s^2 Sin\beta . Cos\beta = mu_s(\dot{\psi} + \dot{\beta}) \qquad (23)$$

Using the torque equations and dividing (22) and (23) by m and mu_s correspondingly yields:

$$\begin{cases} \dot{u}_s = C_x u_s^2 Cos^2\beta + C_y u_s^2 Sin^2\beta - \dfrac{k_{mr_x}}{m} T_{mr} Cos\beta \\ \dot{\beta} = -r - C_x u_s Cos\beta\, Sin\beta + C_y u_s Sin\beta . Cos\beta + \dfrac{k_{mr_x} T_{mr} Sin\beta}{mu_s} \\ \dot{r} = \dfrac{-l_{tr}}{I_{zz}} T_{tr} \\ \dot{\psi} = r \end{cases} \qquad (24)$$

Where, the last equation in (24) comes from the Euler angle equations. For the positions in the global inertia coordinate we have:

$$\begin{cases} \dot{x}_c = u.Cos(\beta + \psi) \\ \dot{y}_c = u.Sin(\beta + \psi) \end{cases} \qquad (25)$$

Then we can see from (25) that if we impose the desired trajectory, although again we have 2 equations and 3 unknowns, at least forward velocity can be computed explicitly and we have the summation of the side-slip and yaw angles. In the next section, this model will be verified by numerical simulations.

1.1.2.1. Numerical Verification of the stability dynamic model

In the previous section, the helicopter 3DOF dynamics is derived in the stability coordinate; to verify the new dynamics, (24) and (25) is simulated for different kinds of inputs. Then, the output of the stability dynamics is compared with the output of the body dynamics (6) and (8). Using Matlab software, the simulation is done and m-file associated with this simulation is "*model_verification.m*". The simulation results for two kinds of inputs are depicted below:

1- For a constant step input:

The constant inputs $T_{mr} = 5$ and $T_{tr} = 0.1$ is applied to both body and stability dynamics and the outputs are depicted in figure 4:

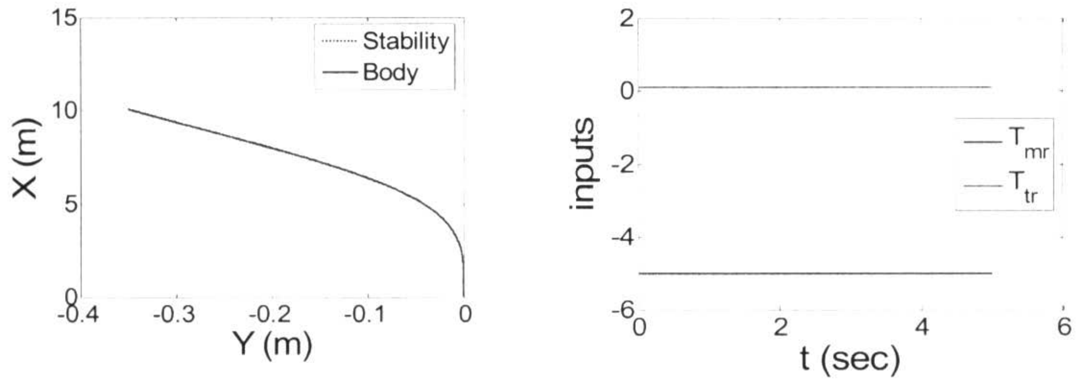


Fig 4. Simulation results of both Stability and body dynamics for constant inputs

2- For a periodic input:

In this case a periodic T_{mr} and a constant T_{tr} is applied to both body and stability dynamics; the simulation results are depicted in figure 5:

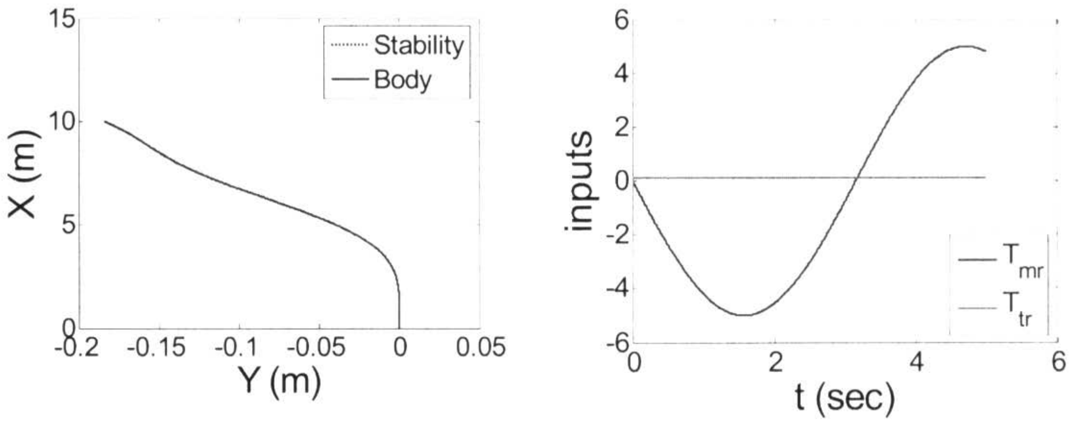


Fig 5. Simulation results of both Stability and body dynamics for periodic inputs

Figure 4 and 5 show that the outputs of both body and stability models are the same when we apply the same inputs to them and their behaviours are similar; this means there is no error in dynamics due to conversion from body coordinate to stability coordinate and the stability dynamics 24 and 25 can be used to design the receding horizon control (RHC) for the 3DOF helicopter.

4.2.4 Quasi-infinite Receding Horizon Control

Suppose that the nominal model is presented as the following general form:

$$\dot{x} = f\big(x(t), u(t)\big) \qquad x(0) = x_0 \tag{26}$$

Where $\mathbf{x}(t) \in \Re^n$ is the state of the system and $\boldsymbol{u}(t) \in \Re^m$ is the input vector satisfying the constraints:

$$u(t) \in U \qquad \forall t \geq 0 \tag{27}$$

U is the set of allowable inputs. In order to meet the necessary conditions for stability, it is also assumed that:

A1- f is twice continuously differentiable.

A2- U is compact and convex and contains origin.

A3- system (26) has a unique solution for a given initial condition [7].

Then, under these circumstances the RHC is the repeated solution of the following problem [9]:

Problem 1: Find

$$\min_{\mathbf{u}(.)} \int_0^T q(x(\tau), u(\tau))d\tau + V(x(T)) \tag{28}$$

Subject to

$$\dot{x}(t) = f\big(x(t), u(t)\big) \qquad x(0) = x_0 \tag{29}$$

$$lb_0 \leq v(x(0), u(0)) \leq ub_0 \tag{30}$$

$$lb_t \leq S(x(t), u(t)) \leq ub_t \tag{31}$$

$$x(T) \in \Omega_r \tag{32}$$

Where

$$\Omega_r = \left\{x \in \Re^n : V(x) \leq r\right\} \tag{33}$$

ν is an initial constraint on the parameters and S is a trajectory constraint enforced over the entire time interval. lb and ub are the lower and upper bounds respectively on the constraints.

Constraint (32) is added to guarantee the stability of the RHC; in fact, by the means of the RHC the trajectory of system is driven to a neighbourhood of the origin and after that a linear local feedback controller stabilizes the system; hence, for the open loop system, control input is as follows:

$$u(s)=\begin{cases} u^*(s;x(t),t) & x\in X\cap\Omega_r^c \\ Kx(s) & x\in\Omega_r \end{cases} \tag{34}$$

Where, x is the set of initial states that there exists a feasible solution for them, Ω_r^c is the set of states which do not belong to Ω_r and K is the gain of local feedback controller.

Remark: in the quasi-infinite RHC the local feedback controller is used in offline computations to prove the stability of the RHC scheme and design the cost function penalties; in that, in the online computations only the RHC controller scheme is used.

Repeating this computation yields a closed-loop feedback control law. For a receding horizon sampling period δ when δ belongs to (0, T]; then, the closed-loop system is represented as:

$$\begin{aligned} &\dot{x}(\tau)=f\left(x(\tau),u^*(\tau)\right) \\ &u^*(\tau)=u^*(\tau;x(t)) \qquad \tau\in[t,t+\delta),\quad 0<\delta\le T \end{aligned} \tag{35}$$

Where $u^*(\tau;x(t))$, $\tau\in[t,t+T]$, is the optimal control obtained from the above problem with the initial condition $\boldsymbol{x(t)}$, and t is the start time of the optimization process and the instant at which states are sampled.

One suitable choice for cost function (28) is the quadratic cost functions, because of their unique properties in the optimization problems [11]. With the following selections, the RHC problem becomes quadratic:

$$q(x(t),u(t))=\frac{1}{2}x^T(t)Qx(t)+\frac{1}{2}u^T(t)Ru(t) \tag{36}$$

$$V(x(T))=\frac{1}{2}x(T)^T Px(T) \tag{37}$$

$Q,P \in \Re^{n\times n}$ and $R \in \Re^{m\times m}$ denote positive-definite, symmetric weighting matrices, T is a finite prediction time.

Since states and control inputs implicitly depend on time, it is required to parameterize the states and control inputs of system as a function of time to solve this optimization problem. The next section discusses the B-Splines as the basis functions for parameterization of states and control inputs.

1.2.1. B-Spline Basis functions

Polynomials are suitable basis functions for trajectory generation since it is easy to evaluate, differentiate and integrate them. However, they have some serious deficiencies; for example, if a function is approximated in a large interval then the order of approximating polynomial can be quite large [10]. Another problem occurs from the global dependence on local parameters; in that, if a fit is poor in a small area then it will be poor in whole curve [10]. In that, if one of the polynomial's coefficients is not set correctly due to an error in curve-fitting, since this coefficient will be used for calculating the curve over all points, then the error will affect all the curve fit?; one remedy is to use B-Spline basis functions.

The general idea in construction of B-Spline curves is to control the trajectory by some control points P_i; figure 6 shows the effect of a control point on the trajectory. The control points attract the trajectory; the value of each control point relative to the value of other control points specifies how much that control point can attract the trajectory.

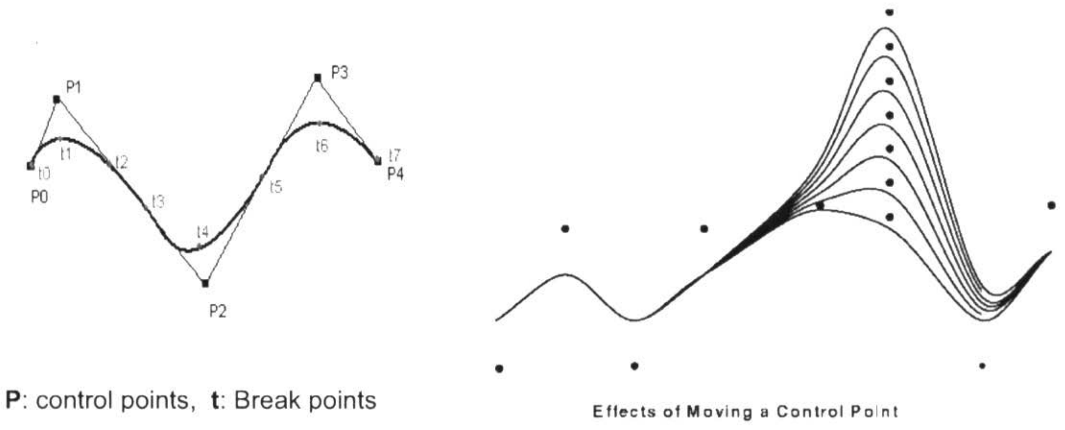

P1
P3
t0
t1
t2
t3
t4
t5
t6
t7
P0
P2
P4
P: control points, t: Break points
Effects of Moving a Control Point

Fig. 6. B-Spline and the effect of control points

Some benefits of B-Spline basis functions motivate engineers to use them for the optimization based trajectory generation; for example:

1- B-Spline offers smoother trajectories than other kinds of basis functions like polynomials.

2- Since B-Splines are constructed by joining several polynomials with a prescribed level of continuity, they are differentially continuous in the break points. Hence, we don't need to check the continuity during the optimization process; and this will reduce the computation burden which is useful for the on-line applications.

3- The trajectory always stays within the area specified by the control points and won't exceed the maximum and minimum control point. Hence, this property can be utilized to apply the constraints in the optimization problem easily. It is important to mention that the number of control points is calculated by equation (40) and the value of each control points is determined by solving the optimization problem for path planning.

B-Spline is a generalization of polynomial curves. It is constructed by joining several polynomial curves with a prescribed level of continuity between them. The points at which the curves are joined are called break points. A non-decreasing sequence of real numbers containing the breakpoints is required to construct the B-Splines and they are called a knot vector. The number of times a break point occurs in a knot vector is called the multiplicity m_i of that break point. The smoothness s_i of a breakpoint provides the level of continuity at a breakpoint. Figure (7) shows the effect of multiplicity on the smoothness of the B-Spline curves; as it is shown for high multiplicity, the B-Spline curve is smoother in the break points.

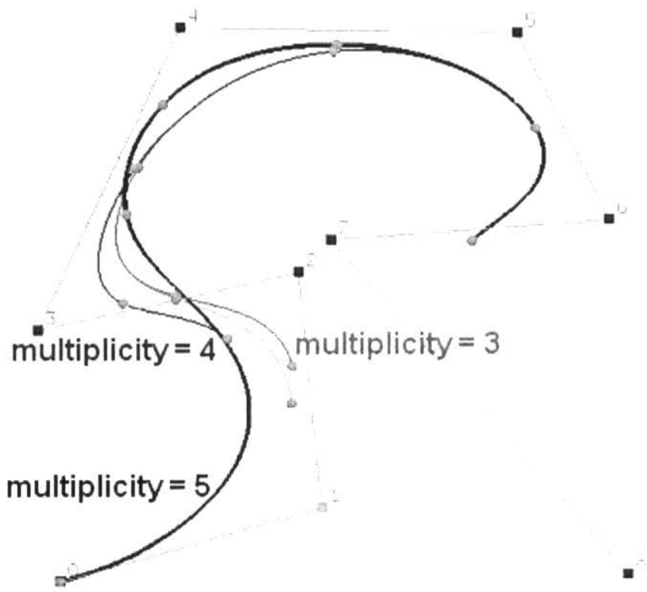


Fig. 7. B-Spline and the effect of multiplicity

A breakpoint is s_i-*1* times continuously differentiable. The order r_i of the piecewise polynomial can be determined from s_i and m_i for interior breakpoints as:

$$r_i = s_i + m_i \tag{38}$$

The equation of a B-Spline with prescribed smoothness s and order r can be written as:

$$y(t) = \sum_{j=1}^{n} N_{j,r}(t) P_j \quad ; \qquad t_1 \le t \le t_m \tag{39}$$

Where, m is the size of knots vector and:

$$\mathrm{n} = \mathrm{m} - \mathrm{r} - 1 \tag{40}$$

Also, $N_{j,r}(t)$ is the basis functions at time t, and P_j are the control points. Mathematically, $N_{j,r}$ is computed from the following recursive formula:

$$\begin{aligned} N_{j,1}(t) &= \begin{cases} 1 & t_j \le t < t_{j+1} \\ 0 & \mathit{otherwise} \end{cases} \\ N_{i,r}(t) &= \frac{t - t_i}{t_{i+r} - t_i} N_{i,r-1}(t) + \frac{t_{i+r+1} - t}{t_{i+r+1} - t_{i+1}} N_{i+1,r-1}(t) \end{aligned} \tag{41}$$

Interested readers are referred to [10] for more information on B-splines.

An m-file is created to calculate $N_{j,r}(t)$ form the above recursive procedure; this m-file is called "*spline_fun_1.m*". This code is a function that calculates $N_{j,r}(t)$ and its 1st and 2nd derivatives at a given time; the inputs are order of B-spline and a given time at which we want to calculate $N_{j,r}(t)$. The control points P_i are obtained by solving the optimization problem.

4.2.5 Implementation of RHC to inner loop controller design

One of the most efficient tools for decreasing the computation effort in optimization problems is the flat output. The main idea is to reduce the dimension of the optimal control problem to a lower dimension easing the formulation and reducing the amount of required computations for optimization process. The motivations for using flat outputs in optimization problems are discussed completely in the previous reports [12] and interested readers are referred to [11] for more detail information and application of flat outputs in the trajectory generation and optimization problems.

The 3DOF helicopter dynamics in stability coordinate is presented in (24); to apply RHC to this dynamics, 3 flat outputs are required:

$$\begin{cases} z_1 = x_1 = u \\ z_2 = x_2 = \beta \\ z_3 = x_4 = \psi \end{cases} \tag{42}$$

Then, all other states and control points can be expressed as a function of the flat outputs and their derivatives. Each of these flat outputs is parameterized by a B-Spline. The order of B-Spline is 3, and the break points are the initial time and finite horizon time (*[0, T]*). It is obvious that with larger prediction horizon time *T*, the solution of RHC will get closer to the optimal solution; on the other hand, larger *T* will increase the computational burden. For larger T, we must define more intervals; hence this will increase the number of break points which requires more control points over a wider interval. The optimization problem then has to be solved over a wider interval; consequently it will increase the computation burden. Hence, selection of *T* is a trade off between computation effort and optimality. For our problem by a trail and error procedure the optimal finite horizon time is chosen as *T=1* sec.

Also, the execution time δ should be as small as possible to meet the necessary conditions for stability and performance; however, in practice, it is limited by the sampling time and the computation time required for running the optimization. Actually, the execution time has to be larger than the sampling time to allow for the communication among agents and measurements by sensors. Also, it has to be larger than the computation time to allow the optimization process be done completely.

In our case, the execution time is chosen as $\delta = 0.1\,\mathrm{sec}$; in fact, it is assumed that the sampling time and the required time for optimization are smaller than 0.1 sec. The matrix penalties of quadratic cost function are chosen as below:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$
$$R = 0.0000001 \times I \tag{43}$$
$$P = \begin{bmatrix} 0.00031 & 0 & 0 & 0 \\ 0 & 13.1705 & 0.00000236 & 132.4 \\ 0 & 0.00000236 & 0.00000307 & 0.000055 \\ 0 & 132.4 & 0.000055 & 1342.795 \end{bmatrix}$$

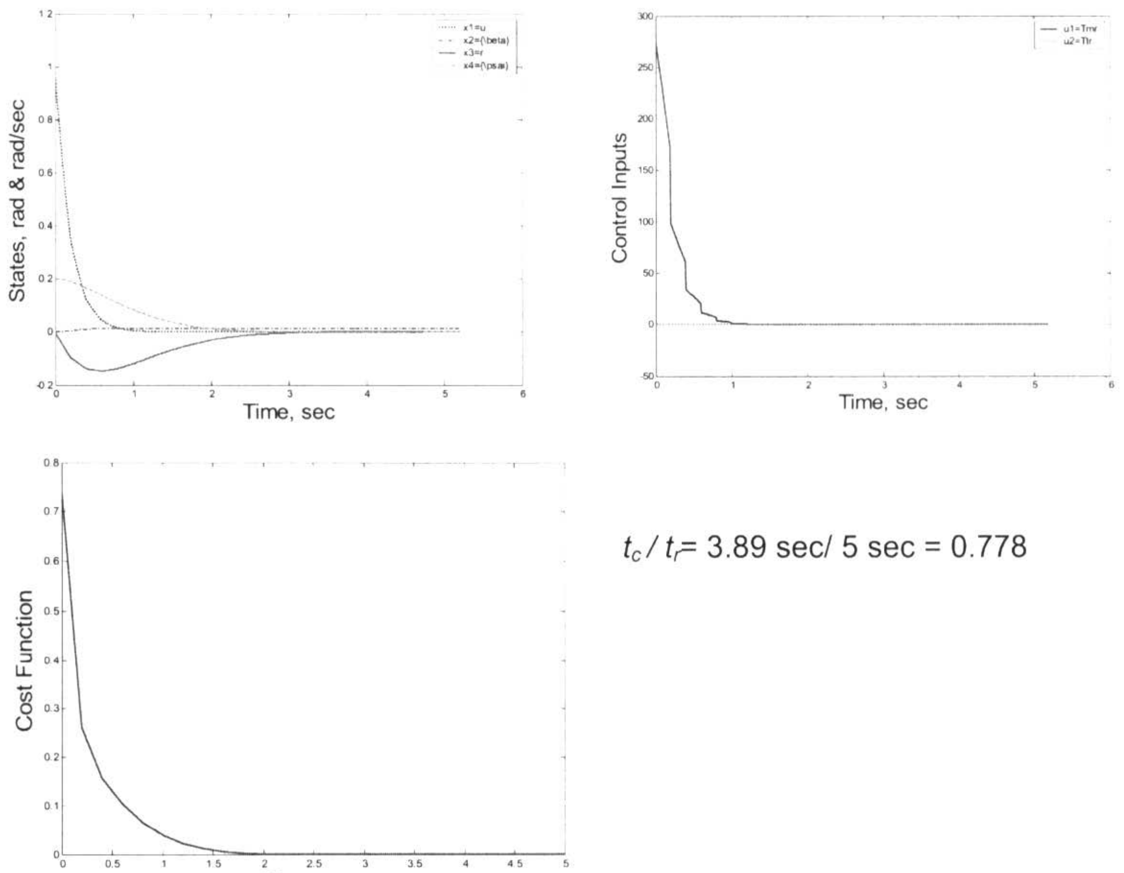
Where, I is the identity matrix. P is obtained from the Lyapunov equation for the linerized of helicopter dynamics, and also, Q and R are tuned based on a trial and error procedure for the best response of system.

4.2.6 Simulation Results:

In this section, the simulation results of the RHC controller designed in the previous section is presented for different kinds of manoeuvres. In all cases, since our simulations are not real time, to investigate the feasibility of the controller in the real world, the computation time, t_c (optimization) is compared to the simulation time or the time required in the real world to do the manoeuvre (t_r). For example, if an aircraft aims to do a turn manoeuvre (following a circle) with a turning rate of 2 deg/sec, it will take 180 sec (t_r), but if we want to simulate this manoeuvre by computer this may take less than 180 sec or more (t_c). For the feasibility, it is desired that the optimization time be less than the manoeuvre time ($t_c < t_r$). The simulation is done by Matlab software and the related code is "*RHC_3DOF_Heli_closeloop.m*" that invokes two other functions called: "*jfun_closeloop.m*" and "*spline_fun_1.m*". The first function represents the cost function to be minimized by RHC and the second one outputs the B-Spline basis function and its 1st and 2nd derivatives to be used for trajectory generation. Complementary comments within the m-files help the user to understand each part of the code. The optimization toolbox of Matlab is used to solve the optimization problem of RHC; this toolbox can solve the optimization problems with all kind of constraints including linear and nonlinear constraints. However, this toolbox cannot be used for real-time implementation; thus, in the future, RT optimization codes such as SNOPT will be used and thoroughly investigated.

1.4.1. Stabilization:

For the stabilization case, the simulation results are depicted in figure 8; the simulation is done for an initial disturbance in forward velocity and yaw angle. As it is illustrated from figure 8, the RHC controller stabilizes the system properly. Also, the simulation time is $t_r = 5$ sec, and the computation time for this case is $t_c = 3.89$ sec which is smaller than the t_r, this means that the simulation can be done in real time applications and the controller is feasible. However, it is an estimate of the computation time for the entire simulation. At each sampling instant the computation time may take longer than the selected sampling period.



$t_c / t_r = 3.89$ sec/ 5 sec = 0.778

Fig8.Simulation results of RHC, inner loop control for stabilizing case

4.2.6.1 Turn over manoeuvre with radius = 100 m

For the tracking case, different manoeuvres are simulated to show the effectiveness of the quasi-infinite RHC method. In the first case, a turn over manoeuvre with radius of 100 m and angular velocity (yaw rate) of $\omega = 0.1\ rad/Sec$ is considered and the simulation results are depicted in figure 9. As it is obvious from figure 9, the RHC controller follows the command trajectory properly: The error in x and y positions is also depicted in figure 9; it is seen that the error in y direction is more than the error in x direction; this is due to the fact that the helicopter dynamics is under-actuated and in fact there is no side controller.

Furthermore, the simulation time in this case is $t_r = 60$ sec, and the computation time for this case is $t_c = 42.438$ sec which is smaller than the t_r; hence, the simulation can be done in real time applications and the controller is feasible.

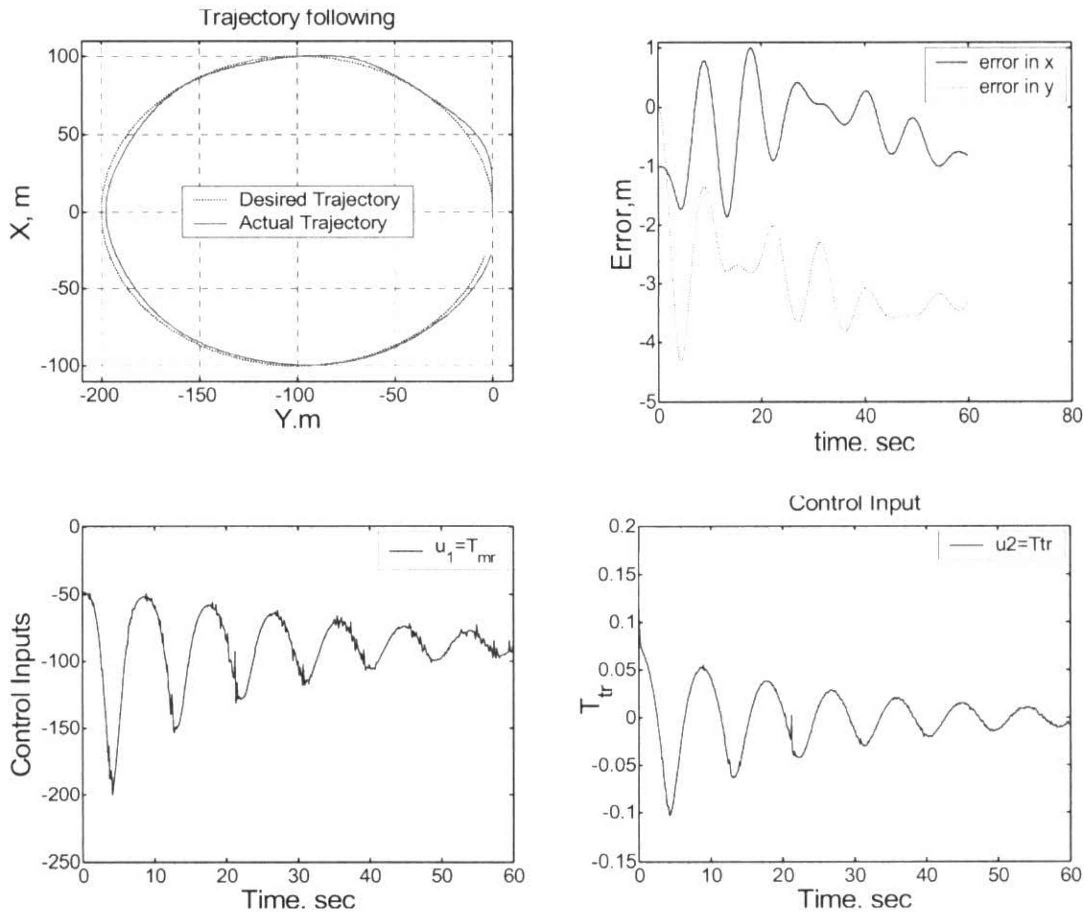


Fig9.Simulation results of RHC, inner loop control for Trajectory tracking (Radius =100m)

1.4.3. Turn over manoeuvre with radius = 25 m

The second tracking case is a turn over manoeuvre with radius of 25 m and angular velocity (yaw rate) of $\omega = 0.05$ *rad/Sec* ; the simulation results are depicted in figure 10. As it is obvious from figure 10, the RHC controller follows the command trajectory properly. The error in x and y positions is also depicted in figure 10; again in this case, although the helicopter is under-actuated, the error is in an acceptable range.

Moreover, the simulation time in this case is $t_r = 120$ sec , and the computation time for this case is $t_c = 89.203$ sec which is smaller than the t_r ; hence, the simulation can be done in real time applications and the controller is feasible.

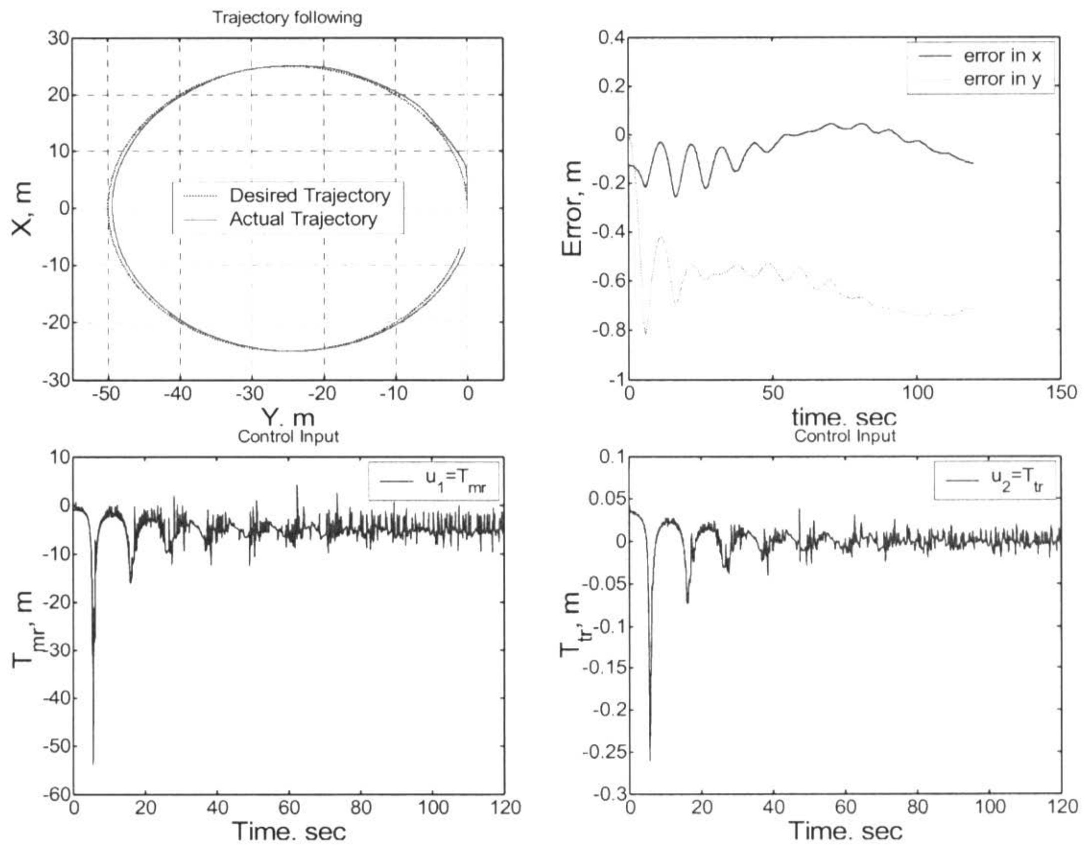
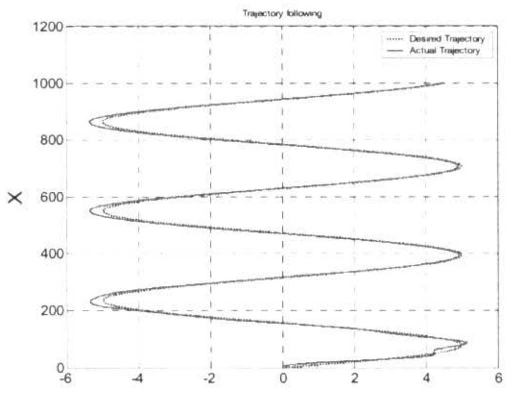


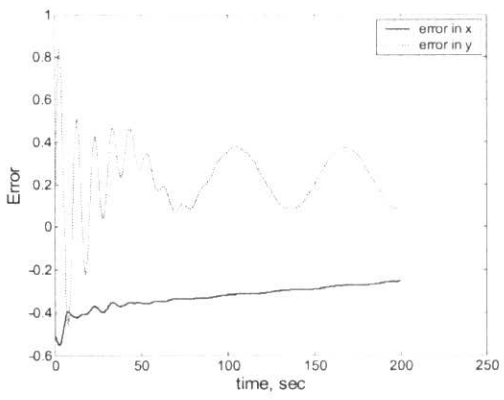
Fig10.Simulation results of RHC, inner loop control for Trajectory tracking (Radius = 25 m)

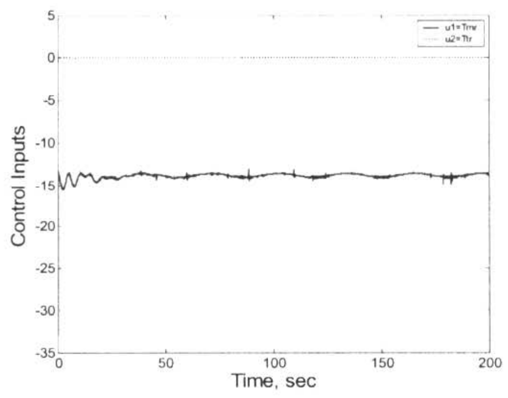
1.4.4. 2D Spiral manoeuvre

In the third tracking case a two dimensional spiral manoeuvre is considered and the simulation results are depicted in figure 11. As it is shown in figure 11, the RHC controller follows the commanded trajectory properly. The error in x and y positions is also depicted in figure 11; again in this case, although the helicopter is under-actuated, the error is in an acceptable range.

Moreover, the simulation time in this case is $t_r = 200$ sec, and the computation time for this case is $t_c = 161.76$sec which is smaller than the t_r; hence, the simulation can be done in real time applications and the controller is feasible.







t_c / t = 161.76 sec/ 200 sec = 0.808

Fig11.Simulation results of RHC, inner loop control for Trajectory tracking

Simulation results show that although the system is under actuated, the RHC controller can control it properly in different situations.

4.3 Decentralized RHC and Problem Scenarios

In the path planning of multi-UAVs a trade-off between "Decentralization" and "Cooperation" modes should be adopted to achieve the best performance of the path planner. However, in some cases, depending on the availability of the communicated information only one of these modes can be used. A fully-decentralized path planner doesn't use the communicated information from other UAVs and plans the path based on an estimate of other UAVs information (such as position). And also, in a fully-centralized path planner it is assumed that there is no restriction on the communication and the UAVs can receive the communicated information whenever they want with any arbitrary sampling rate. In fact, the path planner doesn't need to estimate the other UAVs information because it can obtain all information through communication with other UAVs.

Depending on whether the path planner uses the communicated information (cooperation) or the predicted information (decentralization), the following architectures for path planning can be proposed:

1- Fully-decentralized path planner (no communication)
2- Overlapping decentralized path planner
3- Overlapping decentralized path planner with coordinator
4- Fully-centralized path planner (only communication)

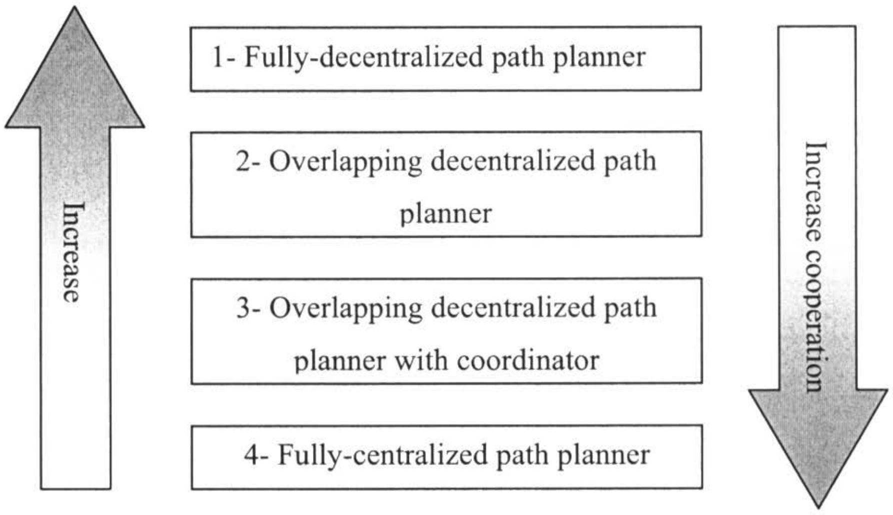

Increase
1- Fully-decentralized path planner
2- Overlapping decentralized path planner
3- Overlapping decentralized path planner with coordinator
4- Fully-centralized path planner
Increase cooperation

Fig. 12: the effect of information sharing the autonomy of each UAV

Figure 12 shows how the autonomy and cooperation will vary by choosing any of the architectures. Fully-decentralized path planner architecture enables the UAV to be completely autonomous and plan its own path without communication with other UAVs; in a fully-decentralized architecture since UAV doesn't communicate with other UAVs, it needs an estimate of other UAVs position to plan its own path and avoid collision with other UAVs. Also, in a fully-centralized path planner the highest level of cooperation is done among UAVs and the lowest level of autonomy will happen. The second and third architectures are a trade-off between the first and forth architectures. Based on the availability and reliability of the resources of information (communication and estimation), any of these four architectures can be triggered during performing the mission. The word "Overlapping" is used because these architectures use a combination of communicated and predicted information. In this report only the "Fully Decentralized Path Planner" will be addressed completely and the other three path planners will be discussed in the future.

In the control of UAVs in a fully decentralized manner, each UAV plans the paths independently without communication with other UAVs; in fact, there is no central decision maker. The only shared objective is the task allocation, done whenever it is needed; in fact, communication is done after doing every set of tasks by UAVs. For example, suppose that 5 UAVs want to do 15 tasks; firstly 5 closest tasks are assigned to 5 UAVs based on the initial position of UAVs; then all UAVs plan the paths towards their assigned task; after finishing their task the UAVs again communicate with each other and inform other UAVs about their new position; again based on the new position of the UAVs, 5 tasks are assigned to 5 UAVs; this procedure will be repeated till performing all tasks by UAVs. However, it is possible to use the communicated information efficiently for improving the path planning unit performance

In the centralized control method, the computation and decision making is done by a single UAV or a central ground station; hence, if this UAV fails, or gets shot down by an adversary, the other UAVs won't be able to carry on the mission. Another advantage of reduced inter-vehicle communications is to lower the probabilities enemy radars detect the UAVs. These benefits of decentralized control over centralized cooperative control have motivated researchers to pursue effective distributed solutions.

In figure 1, the schematic block diagram of cooperative control for one UAV is shown. The UAV and Controller blocks are discussed and designed in chapter 1. In this chapter, we will work on the Task Assignment and Path Planning blocks.

Each mission scenario consists of two main parts: *task assignment* and *path planning*. The task assignment part is a decision process; in fact, based on the minimization of a cost function, one or more targets will be allocated to each UAV. The cost function can be the sum of the distances between UAVs and targets. In some simple mission scenarios where there are no moving and pop-up targets, the task assignment will be done once at the beginning of scenario and the assigned task won't be updated during mission. Then, some issues such as pop-up targets and threats are referred to the *task assignment* part; it means when some unexpected targets are discovered, the task assignment unit may change the current decision and assigns the discovered target to the closest UAV.

As soon as the targets are allocated to each UAV, the path planning begins. In the path planning block, it is typically assumed that the UAV knows its own initial position, the initial position of some (or all) other UAVs, and the position of the target assigned to itself. Then, based on this primary information, path planning is done under the condition that UAV has to visit the assigned target without colliding to other UAVs. One important question here is that since there is no communication possible among UAVs, how collision avoidance between UAVs is guaranteed during flight? We will answer this question later for our proposed approach. Some problems such as obstacle avoidance are referred to *path planning* part; in fact, the planned paths mustn't pass through the obstacles.

The path planning method must be such that the planned path is updateable during the flight; especially, in the case of moving or pop-up targets; because, in these cases, the task assignment unit will update its decision based on the new position of moving targets or position of pop-up targets.

In this section, we will introduce some of the recently presented scenarios in the open literatures:

1- SEAD (Suppression of Enemy Air Defences) [1]:

The first scenario is discussed in the research works of West Virginia University. A simple battlefield is shown bellow; UAVs are shown as blue diamonds numbered 1 through 4 along the left side of the battlefield. The static targets are green 'x's, while the single pop-up target is shown as a green '+'. The no-fly zones are the obvious black circles. Threats are shown as a red star with surrounding effective radius for the static variety, and the pop-up threat is the large read range with the red 'O' at the center.

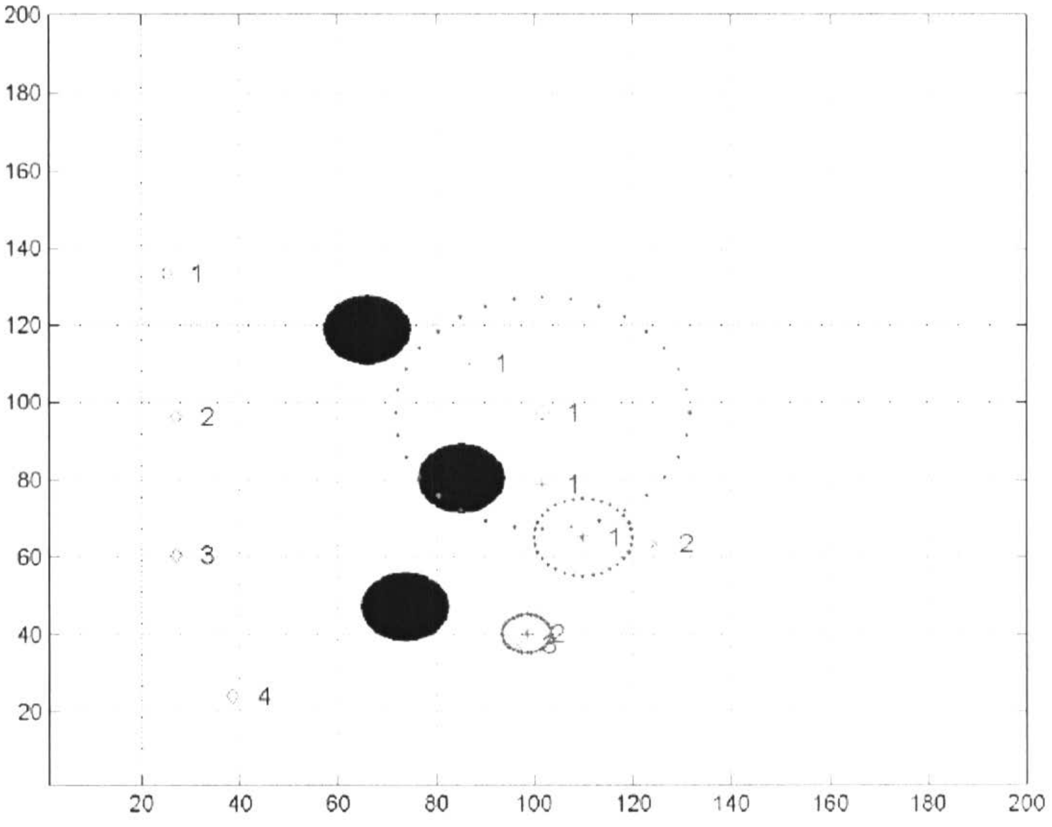


Fig. 13. Suppression of Enemy Air Defences scenario

Given information:

1- *nuav*: number of UAVs
2- *ntarg_s*: number of stationary targets or waypoints to be visited
3- *ntarg_p*: number of pop-up targets
4- *nzones*: number of no-fly zones (obstacles)

Goal: UAVs must visit each target while minimizing an overall cost to the group

2- Cincinnati University [8]:

The second scenario comprises the following elements:

1- N heterogeneous UAVs.
2- M_k stationary targets; M_k are initially known
3- M_h pop-up targets; M_h is unknown a priori.
4- A bounded $Lx \times Ly$ mission environment in which the UAVs operate. The environment is represented in the UAVs' information bases as a grid of cells:

$$\{(x, y), x = 1, \ldots, L_x, y = 1, \ldots, L_y\} \qquad (44)$$

It is assumed that each cell contains at most one target, and that targets do not cross cell boundaries.

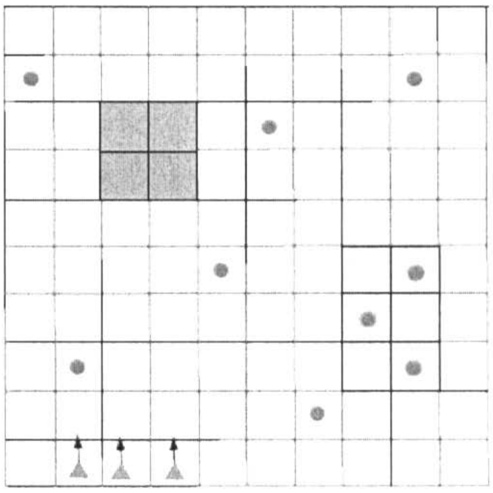


Fig14. A typical multi-UAV Scenario proposed by Cincinnati University

Triangles indicate UAVs and circles are targets; the blue region may denote a lake with a zero target probability and the lighter green region may indicate a camp with a high target probability. The mission of the UAV team is to discover, identify and verifiably destroy all targets in an uncertain environment. The UAVs do not have access to centralized information, and each UAV makes its decisions based on subjective information obtained through observation and communication with other UAVs. However, rather than operating separately, they cooperate in two ways: 1) by sharing information among the team; and 2) by coordinating their tasks.

To perform this scenario, an algorithm is proposed where UAVs autonomously make cooperative task allocation decisions using a common information base, providing them with a global view of the mission. The focus of this approach is on the use of very simple decision-making rules by individual UAVs rather than solving a global optimization problem.

3- The Ohio State University [13]:

The third scenario is defined in the Ohio state university. There is a group of Autonomous Air Vehicles (AAV) that performs surveillance of a given area and can use information from a satellite to pursue suspected locations in that area. The set of AAVs is given a number of targets to search for, with their respective likely locations. AAVs must work together autonomously in order to try to maximize the probability of finding targets in the environment with minimal AAV effort. This cooperation must be performed in spite of imperfect inter-vehicle communications, less than full communication connectivity between vehicles, uncertainty in target locations, and imperfect vehicle search sensors.

They decompose the search area into NQ cells. These cells are numbered so that the discretized search space is $Q = \{1, \ldots, NQ\}$. It is assumed that all the AAVs know how the cells are numbered and hence know NQ. It is also assumed that there are NV AAVs that search for targets and let $V = \{1, \ldots, NV\}$ denote the set of vehicles.

It is assumed that there are ND distinct valid stationary targets that the AAVs are searching for and let $D = \{1, \ldots, ND\}$ denote the set of targets. It is suppose that the size of each target is considerably smaller than the size of each cell. Also, there could be more than one target located in one cell; however, it is assumed that all targets inside the same cell are separated in such a way that they can be detected by the sensors of the AAVs.

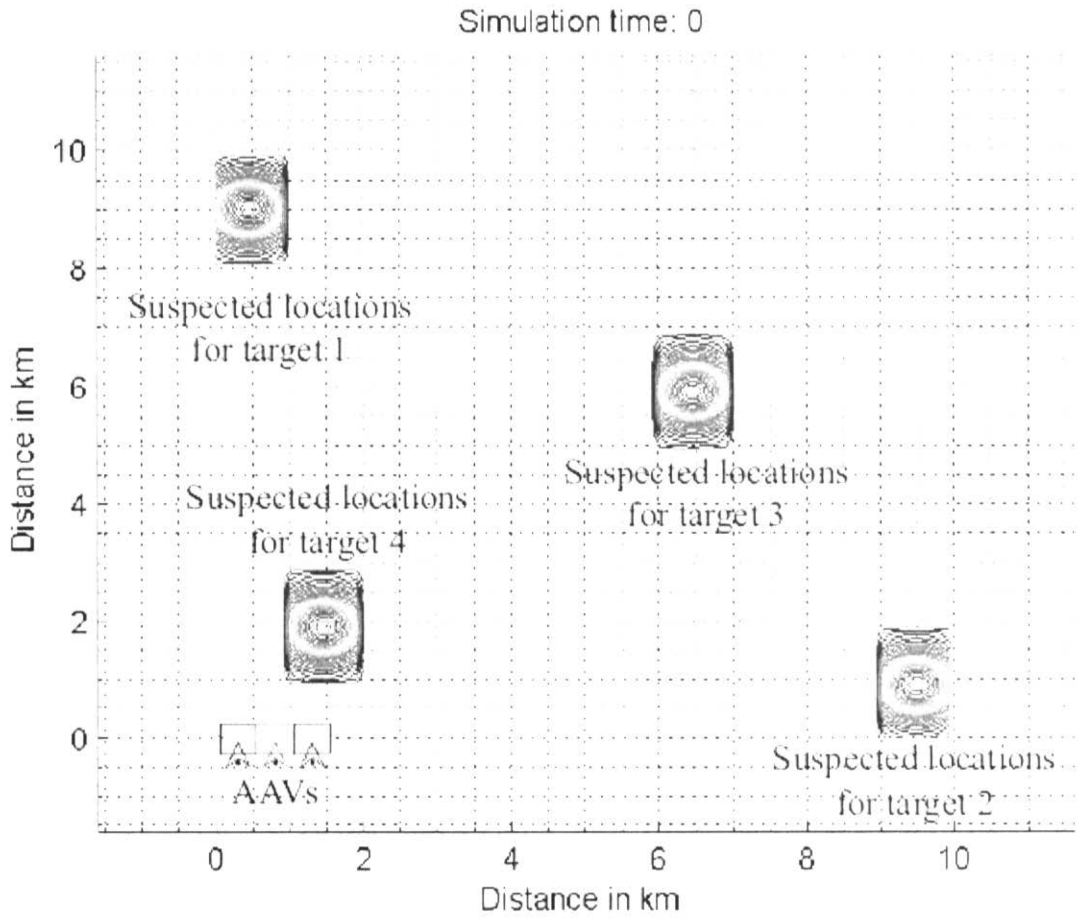


Fig.15. A typical multi-UAV Scenario proposed by Ohio State University

They represent the state of the search progress with a search map and use an invariant set to model the set of states where there is no useful information on target locations. They also show that the invariant set is exponentially stable for a class of cooperative surveillance strategies.

4- University of Leicester, Leicester, UK [14]:

This scenario finds centralized trajectories for 3 UAVs which start from different positions but fly to the same destination. The operational region is 180 km×200 km and has 10 defence units (radar) shown as circles in the figure 16. Five units are of medium range (25 km) centered at coordinates (100,100), (125, 65), (125,135), (50,155), (50, 45), and the rest are of short range (7 km) centered at (42,102), (167,182), (167,127), (167,37), (167,77). The initial positions of UAV1, UAV2, UAV3 are (10, 10), (10,120), (10,180), respectively. All three UAVs start at the same time and move towards a common goal at (170,100). The UAVs fly at an initial speed of

200 m/s and with an initial heading angle of 40° with regard to the horizontal. It is required that the final speed of each vehicle when arriving at the destination is 100 m/s and the heading is along the x-axis; the maximum flight speed is 300 m/s. In this scenario the finite horizon T is chosen as 25 seconds, i.e. , the trajectory is evaluated for five time steps each of 5 seconds duration but only the first control input is implemented.

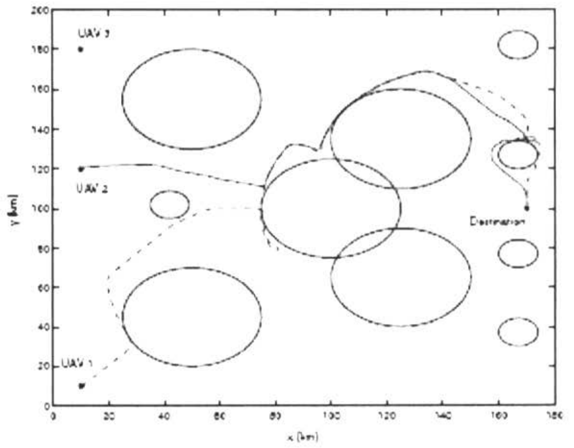


Fig16. A typical multi-UAV Scenario proposed by University of Leicester

The defence units are modeled as squares containing the circular threat regions. The vehicles may enter threat zones but will try to leave them as soon as possible to keep constraints violations at a minimum. It has been observed that maximum violations will occur when the vehicle enters a threat zone at maximum speed and is perpendicular to the square boundary at a tangent point with the circle.

5- MIT Scenario:

In the MIT scenario a group of heterogonous vehicles perform a variety of tasks in dynamic and uncertain environments. In this mission scenario a real-time coordination and control algorithms is designed for teams of multiple UAVs with a focus on balancing performance with improving computational scalability, reducing communication requirements, and increasing robustness to errors in the situational awareness.

A team of autonomous aerial and ground vehicles are coordinated to destroy some pre-specified targets; however, similar to the previous mission scenarios the pop-up targets and threats are included to the scenario that makes it more general and complex [15,16,17, 18].

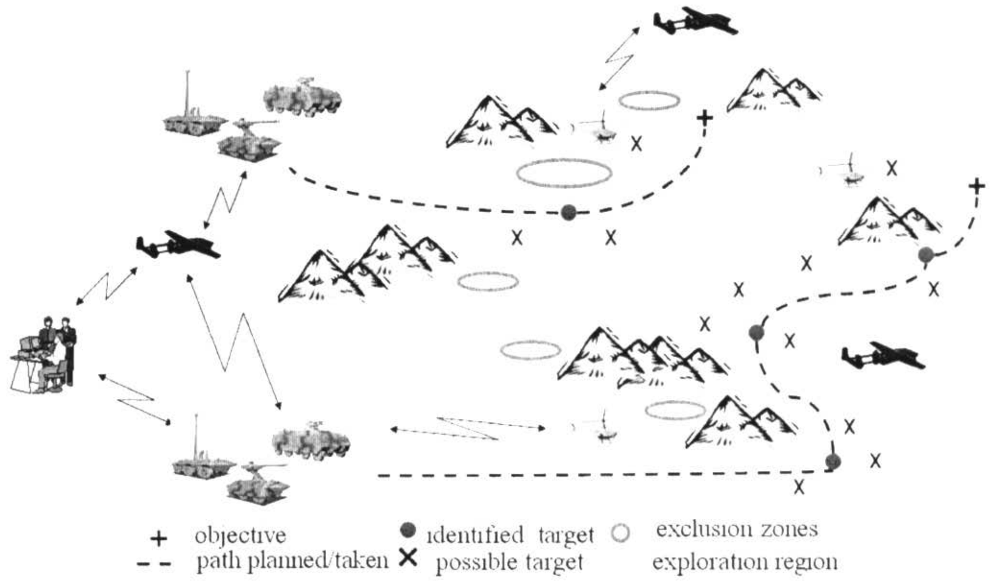


Fig17. A typical multi-agent Scenario proposed by MIT team

4.3.1 Proposed scenario

Each mission scenario consists of two main parts: *task assignment* and *path planning.* For the task assignment part, we assume that there are *N* UAVs, and *N* targets, and UAVs know the initial position of all UAVs and targets; in fact, all UAVs know the initial position on other UAVs and all targets. The task assignment unit will assign one task to each UAV based on the minimization of an overall cost to all UAVs.

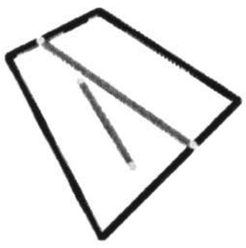
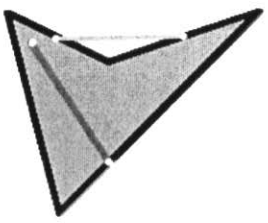
For the path planning part, we assume that *N* targets are allocated to *N* UAVs. UAVs start from some initial positions in their workspace and have to visit their assigned targets without colliding to each other. It is assumed that each UAV initially knows the following information:

Given:

1- *N*: Number of UAVs
2- Initial position of all UAVs
3- *N*: Number of targets
4- Initial position of targets

Objective: UAVs have to visit their assigned target while minimizing an overall cost function.

Remark1: Path planning in a dense environment often gives a non-convex optimization [19]; then, it won't be addressed in this report and will be discussed in the future works. By definition a set S in the Euclidian plane is called a convex set if the segment line joining any two points of S lies in S. For example:

		
a- Convex	b- Not a convex	c- Convex

Hence, if we add an obstacle into a convex set, this will make the set non-convex; this means, the convex optimization methods can not be used for this kind of problems.

4.3.2 Task assignment

Although the main focus of this report is path planning, the task assignment will also be discussed shortly in this section. Because, since the initial position of the UAVs and targets are generated randomly and they are not predefined to be sorted in the best order, a simple task assignment unit is required to allocate the targets to each UAV before the path planning begins; this cause the paths not to intersect each other.

The cost for each UAV to visit a particular target is a function of the path taken. In our proposed approach an overall cost to go for all UAVs is minimized and one target will be assigned to each UAV. In fact, all the possible cases (permutations) of vehicles to targets are considered and a globally optimal permutation would be chosen. The task assignment results for different number of UAVs and targets are shown in figure 18 and the m-file related to this simulation is "*task_assignment.m*".

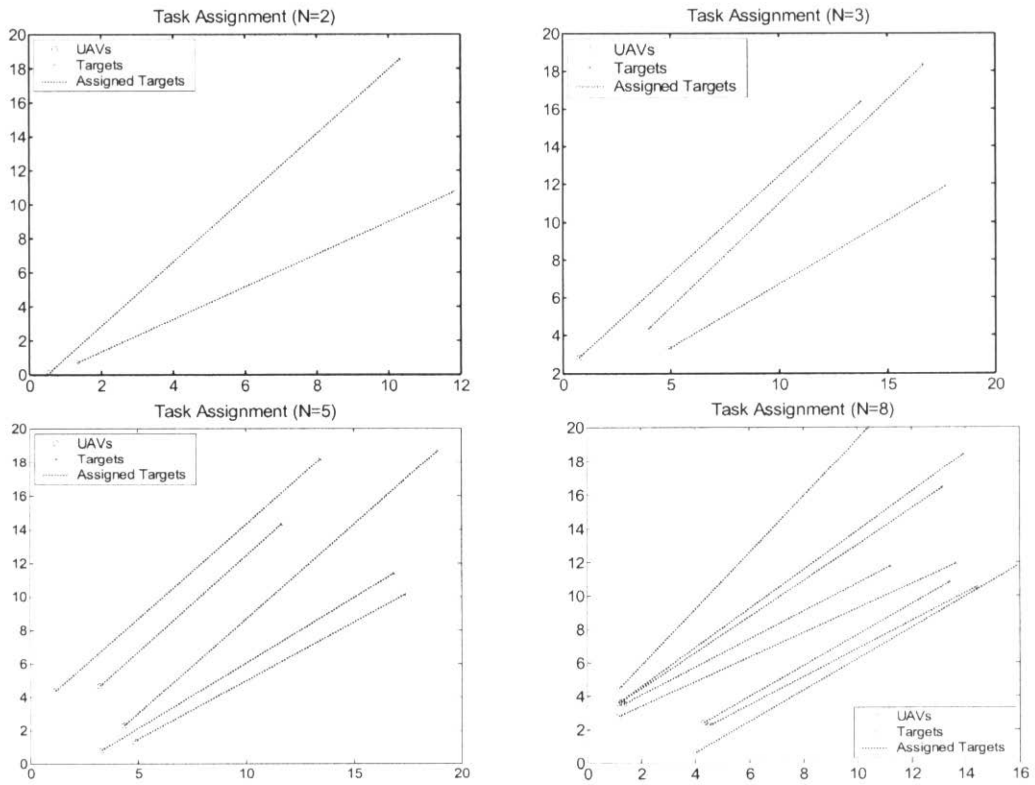


Fig18. Task assignment simulation results for different number of UAVs

4.3.3 Different methods in path planning of UAVs

2.3.1. Graph based method

In the graph based method, paths are generated from a sequence of edges connecting vertices of the graph, optimal control, which computes an optimal path based on a cost function, and finally virtual potential fields, where a simpler, related problem is solved to obtain the path.

In the graph based method, a sequence of vertices is assigned to discrete points in space; then, edges are used to connect these vertices. After that, costs are assigned to each of the edges, and lastly the graph is searched for an optimal trajectory. For a simple graph, vertices can be assigned rectangular points.

2.3.2. Potential function methods

The second method to UAV path planning is virtual potential fields and forces, as proposed by Bortoff [20]. In this method, a chain of masses connected to each other by springs and dampers represents a UAV path. Obstacles to be avoided, such as radar and threats, have repulsive force fields that shape the path until equilibrium is reached. This method has had the smallest amount of research performed among the other methods, though Bortoff concludes that the method is quite promising for uniform radar field [1].

2.3.3. Probabilistic Road maps

Probabilistic roadmap planning (PRM) is an efficient method to compute collision-free paths for UAVs [21]. This method consists of two phases, a building and a query phase. The building phase is the construction of a graph called 'roadmap'. The nodes in the roadmap are collision-free configurations and the edges linking the nodes are collision-free paths. The query phase is finding a path between an initial and goal configurations by connecting these nodes to the road map and searching them for a sequence of edges linking the two nodes. This method was originally developed for holonomic robots in a static environment; however, it has been recently applied to non-holonomic robots with constrained kinematics and high degrees of freedom [22]. In some new extensions of this method, a coarse roadmap is built during the building phase and further refined in the querying phase with focus on the area of interest, and customized to specific preferences such as maximum number of sharp turns [23].

2.3.4. Optimization based methods (RHC)

In this approach a cost function consisting of a path length or time or fuel is minimized to generate an optimal path. The optimization problem is subject to constraints of the starting and final aircraft states and a model of the aircraft; some limitations like maximum speed have to be applied to the optimization problem. The main drawback of this method is that it needs a high computational time; and it increases dramatically with the scale of the problem. Hence, to reduce the computational time a finite horizon time is used instead of an infinite horizon time.

Designing a whole trajectory with a planning horizon fixed at the goal is very difficult to perform in real time because the computational effort required grows rapidly with the problem size. This limitation can be avoided and trajectories in real time can be obtained by using a finite receding horizon approach to form a shorter plan at each simulation step that extends towards the target but does not necessarily reach it.

In this approach, the path is computed online by solving the RHC over a limited horizon at each time step. The procedure is composed of a sequence of locally optimal segments. At each time step, the optimization is done for *T* future time intervals, where the length *T* of the planning horizon is chosen as a function of the available computing resources as well as the individual problem. Solving this optimization provides the input commands for the *T* future time steps. In fact, the path optimization is done for a finite period of time and the first portion of this path is used until the next update is available. Figure 21 shows a schematic representation of this approach.

4.4 Proposed Approach for Decentralized RHC

In this section, a decentralized RHC path planning approach is proposed to design on-line cooperative path planning for a group of UAVs aiming to do the proposed scenario in section 4.3.1. The schematic diagram of a decentralized control for 3 UAVs is depicted below:

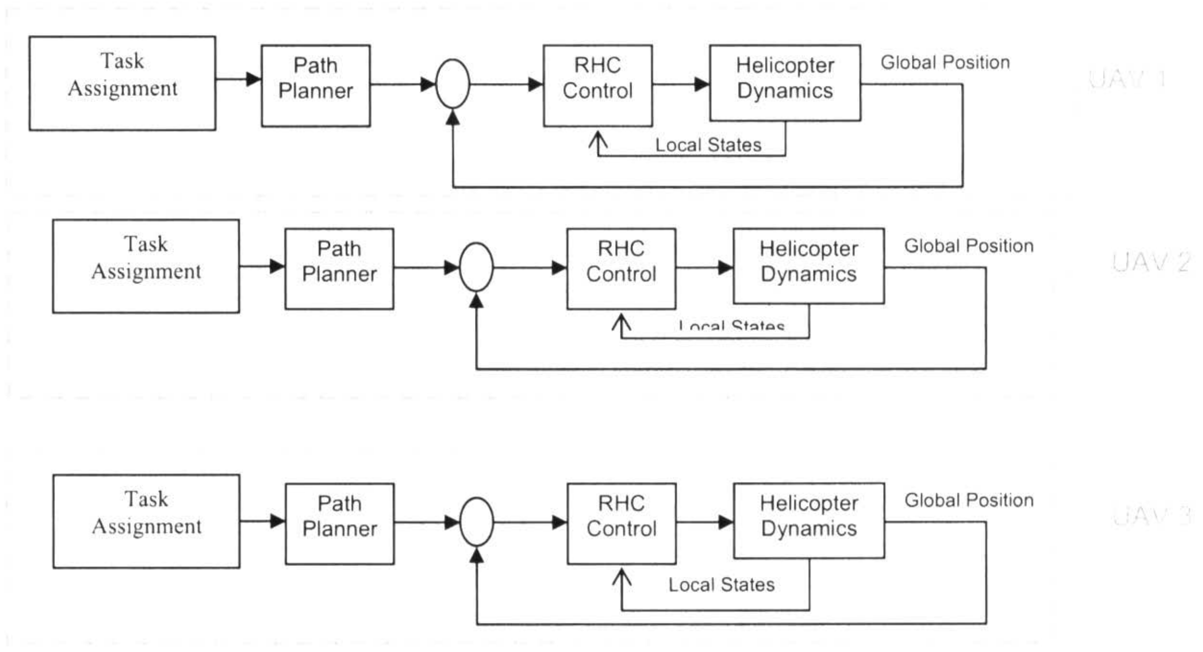


Fig19. Block diagram of decentralized path planning for multi-UAVs

As it is seen from the above diagram, all tasks including task assignment, path planning and control of the UAVs are done decentrally; and the UAVs need no communication during flight.

The only information that UAVs need is the "*initial position*" of other UAVs; this information can be measured by UAVs.

The path planning method is the optimization based (RHC) method introduced in section 4.3.3.4. In this method, an overall cost to all UAVs is minimized to generate the paths; in fact, each UAV 1- plans its own path and 2- estimates the paths planned by other UAVs; these two is obtained by minimization of a cost function by each UAV. The cost function in our case is the distance from the point reached at the end of planned horizon (T) to the target; this cost function will be discussed completely later. A mathematical model of this cost function is used in the path planning unit to be minimized. The minimization of this cost function yields the planned paths and needs no information about the position of other UAVs during the flight. In fact, each UAV minimizes this cost function to plan the paths for all UAVs; but uses the path associated with itself.

The reason beyond the fact that each UAV plans the paths for other UAVs is to estimate the position of other UAVs in order to avoid colliding with other UAVs; this estimation compensates for lack of reliable information in the mode of "Fully Decentralized Path Planning".

One important question is that, how one can ensure that all UAVs make the same decision in different situations? How is it assured that there is no difference between the estimated paths and the paths followed by other UAVs? How the uniqueness of the decisions is guaranteed? The optimal nature of the RHC answers this question. More specifically, since the global minimum of the optimal control is unique, if all agents achieve the global minimum of an overall cost to go, it can be concluded that they will make the same decisions and they need no communication.

The global minimization and achieving the global minimum do not need the exchange of data while the mission evolves; since the initial condition and final conditions (target positions) are known a priori by UAVs; then the optimization problem has a unique global minimum. For example, when in an optimization problem two points are given, the path with minimum distance between these two points is the straight line between two points and this line is unique; hence, if this optimization problem is solved by several processors independently, all of them will get the same answer, provided that they achieve the global minimum of the optimization problem)

The inner loop controller (figure 19) is designed and discussed in section 4.2. and also the task assignment is developed in section 4.3.2. In this section, the path planning unit is discussed and finally a decentralized RHC path planner will be designed.

4.4.1 Important Issues in Path Planning

Some of the most important issues in the path planning of multi-UAVs are:

1- The paths have to satisfy the initial condition of UAVs.

2- The paths have to visit the target positions.

3- The paths must be designed so that the UAVs don't collide each other.

4- The paths must be feasible (it has to satisfy maximum acceleration, speed and tuning rate of UAVs).

The first and second issues are easy to satisfy; all kinds of basis functions can easily apply the initial and final conditions. For example, in our case, when we use B-Spline basis functions for trajectory generation, it is enough to set the fist control point to be equal to the initial condition for applying the initial condition. Then, we will discuss the 3rd and 4th issues in this section:

3.1.1 Collision avoidance among *UAVs*

The collision avoidance is a challenging problem in the cooperative control field. In [17], collision avoidance between the UAVs is dealt with in a way similar to the case of stationary obstacles. At each time step every pair of vehicles must be a minimum distance apart from each other in all coordinates. On the other hand, since the distance of two UAVs is a nonlinear second order function of their positions, applying this kind of nonlinear constraint to the optimization problem is demanding and time consuming; especially in situations where the number of UAVs is huge. Moreover, treating the neighbour UAVs like an obstacle will make the optimization problem non-convex.

One alternative method is the "potential function method"; this method has been found the most attraction because of their efficiency and low computation time.

In the potential field method, the idea is to fill up the UAV's workspace with an artificial potential field in which the UAV is attracted to the target and is repulsed away from the obstacles and other UAVs. One general form of these functions is:

$$J_i = \int_{t_0}^{t_f} \sum_{l \neq i}^{N} \frac{K_{il}}{a_{il}(x_i - x_l)^2 + b_{il}(y_i - y_l)^2} \tag{45}$$

Where x and y are components of position and the degree of importance of each of them is specified by a and b. The above potential function forms a cost function for UAV i, and the importance of this cost function is specified by matrix K.

3.1.2. Linearization of nonlinear constraints

The forth issue in the path planning of the UAVs is the feasibility of the generated paths. By the feasibility of the paths, we mean that since the trajectories are parameterized in time, they have to satisfy the maximum speed and acceleration of the UAVs. Again, like the distance the overall velocity of the UAV is a nonlinear function of component velocities:

$$V_{total} = \sqrt{V_x^2 + V_y^2} \tag{46}$$

Although the exact representation of this constraint would be nonlinear, they can be approximated by linear inequalities. The true magnitude constraints enclose a circle on the *X-Y* plane, as shown in figure 20:

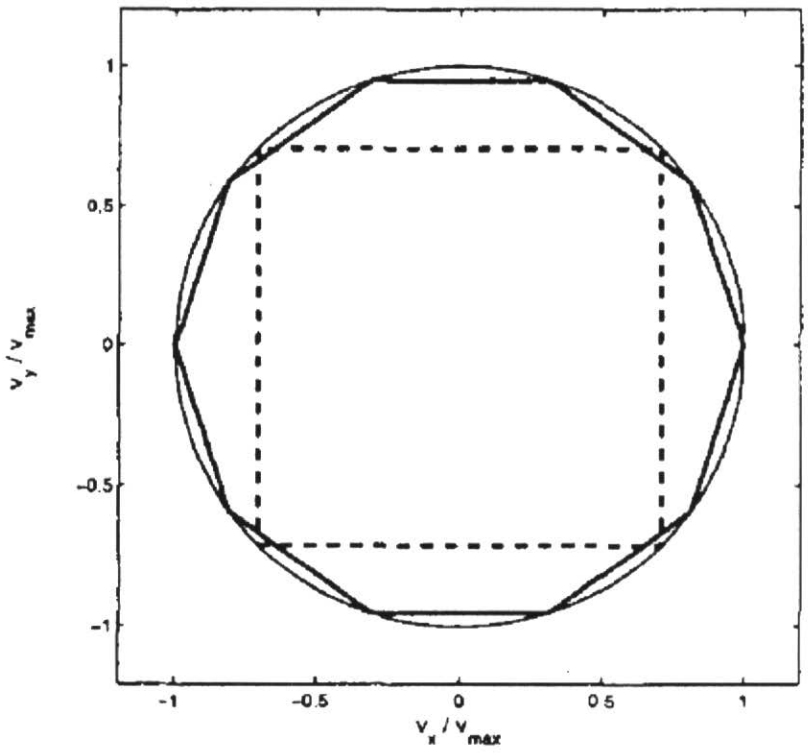


Fig20. Approximation of circle by polygon for linearization of nonlinear constraints

This circle can be approximated by a surrounded polygon; hence, the linear representation of the nonlinear constraint of velocity is:

$$\frac{v_x}{V_{\max}} \sin\left(\frac{2\pi m}{M}\right) + \frac{v_y}{V_{\max}} \cos\left(\frac{2\pi m}{M}\right) \le 1 \times \cos\left(\frac{\pi}{M}\right) \qquad m = 1 : M \tag{47}$$

Where, *M* is the degree of polygon; for *M=4* and *M=10*, the polygons are depicted in figure 20. Simulation results show that *M=12* can construct a good approximation to the circle.

4.4.2 Proposed RHC Based Path Planning Method

Figure 21 shows the way that the path is generated by the means of RHC:

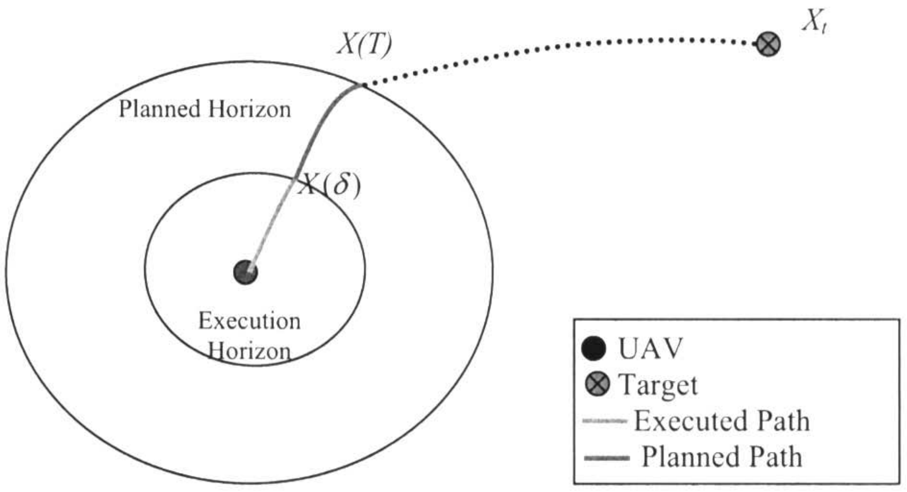


Fig21. RHC based path planning

At each time step, path is planned for a finite horizon time *T* towards the target and does not necessarily reach the target; the first portion of this path is executed during the execution time δ. In figure 21, the orange path is the executed portion and the green one is the planned path. At the next time step, the path planning begins from $X(\delta)$ towards the target.

Since in the RHC scheme the future path is updated after every execution time, it can be very flexible to moving targets and obstacles and even pop-up targets.

In the RHC based path planning, the idea is to minimize a cost function for generating the paths; the type of cost function is very important for the optimization problem. One choice for the cost function is the distance between the *X(T)*, the position at finite horizon time, and X_t, the target position [24]:

$$J_i = d(X(T), X_t) = (X(T) - X_t)^T P(X(T) - X_t) \qquad (48)$$

Where, *P* is a positive definite matrix which is typically the identity matrix, $X(T)$ is the vector of position at finite horizon time and X_t is the position of target (figure 21).

Hence, in our proposed method, the overall cost function is the summation of (48) and (45) cost functions; the first one is used to generate the trajectory and the second one is used to handle the distances between UAVs. Then, the optimization problem for i^{th} UAV is formulated as follows:

Optimization Problem 1:

Minimize:

$$\underset{X^{*}(t)}{Min}\ J_i = (X(T) - X_t)^T P(X(T) - X_t) + \int_{t_0}^{t_f} \sum_{l \neq i}^{N} \frac{K_{il}}{a_{il}(X_i - X_l)^2} \tag{49}$$

Subject to: constraints on the UAV dynamics (Speed and Acceleration)

Where X is the vector of position for all UAVs and X_i is the vector of position for i^{th} UAV. N indicates the number of UAVs and P, K and a are matrix penalties.

As it is obvious, the information of all UAVs appear in the cost function; therefore, since this architecture is fully decentralized, the information of the other UAVs is predicted by the means of RHC based path planner and no communication is required during performing the mission; the architecture only needs the initial position of vehicles and targets; it is assumed that this information is sent to all vehicles before starting the mission.

All UAVs solve the above optimization problem; solving the above optimization problem yields the planned paths for each UAV and an estimate of other UAVs paths. Each UAV follows its own planned path and keep the "estimate of other UAVs paths" in a buffer to use that for future estimation of other UAVs path. The following will propose an algorithm for implementation of this RHC-based path planner.

To develop the idea some parameters should be defined. Suppose that it is desired to plan the paths for the time interval $[t_0, t_f]$; in fact, at time t_0, the UAVs are at their initial position and aim to visit the targets at time t_f. The prediction horizon is shown by T. t_c indicates the computation time for each prediction horizon and δ shows the execution time. t represents the current time and the index "k" is used to show the time step. The following diagram shows the schematic representation of these time notations for i^{th} UAV in the scheduling diagram of RHC based path planner; this timing sequence is representative of on-line and on-board processing. The computations take place during t_c, for each UAV by UAV itself. All UAVs use this timing sequence for the RHC-based path planner:

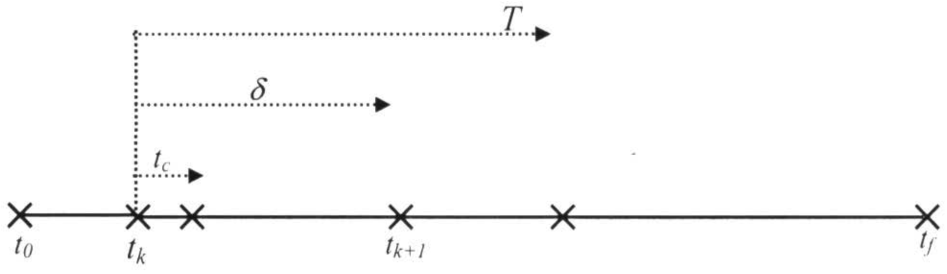


Fig. 22: Schematic diagram for fully decentralized RHC based path planner without Communication

This diagram represents the timing sequence of the RHC-based decentralized path planner excluding inter-vehicle communication. For simplicity in this section, we assume that the computation time here is zero. Taking into account that the initial position of each UAV and also the position of target assigned to each UAV are known to the rest of the group at time t_0, the algorithm of path planning for i^{th} UAV is summarized as follows based on the timing sequence of figure 22:

Algorithm1:

1- Get the position of other UAVs at time $t=t_0$, k=0 (from sensing or using the data from ground control station).

2- Get the position of targets and information on which targets assigned to each UAV at time $t=t_0$ (from sensing or using the data from ground control station).

4- Plan the path and estimate the paths of other UAVs for time interval $[t_k, t_k+T]$ by solving the optimization problem 1.

5- Command the planned path over the interval $[t_k, t_{k+1}]$ to be followed by i^{th} UAV.

6- Sample the position X_{actual}, of i^{th} UAV at $t=t_{k+1}$. (X is the vector of position).

7- Calculate the position $X_{estimated}$, of other UAVs at $t=t_{k+1}$, using the estimated paths of step 4.

8- $k=k+1$. Goto step 4.

This procedure is repeated till visiting the target position. The next section will address the improved algorithm. The simulation results for this RHC based path planner will be presented in the next section.

4.4.3 Simulation Results

In the RHC based method for path planning discussed in the previous sections, the cost function is the summation of (48) and (45) cost functions. Also, the method of section 3.1.2 is used for linearization of the maximum velocity constraints. In this section, this method is simulated by Matlab software for different number of UAVs and the related m-file is "*Path_Planning.m*" which invokes three other functions called: "*task_alloc.m*", "*cost_to_go_2.m*" and "*spline_fun_1.m*". "*task_alloc.m*" is a function that allocates the targets to each UAV; "*cost_to_go_2.m*" is a function that forms the cost to go for all UAVs form the position at the finite horizon time to the targets position and "*spline_fun_1.m*" outputs the B-Spline basis function and its 1st and 2nd derivatives. B-Spline basis functions of degree 3 are used to generate the paths. The following figures show the simulation results for a finite time horizon *T=5 Sec* and different number of UAVs:

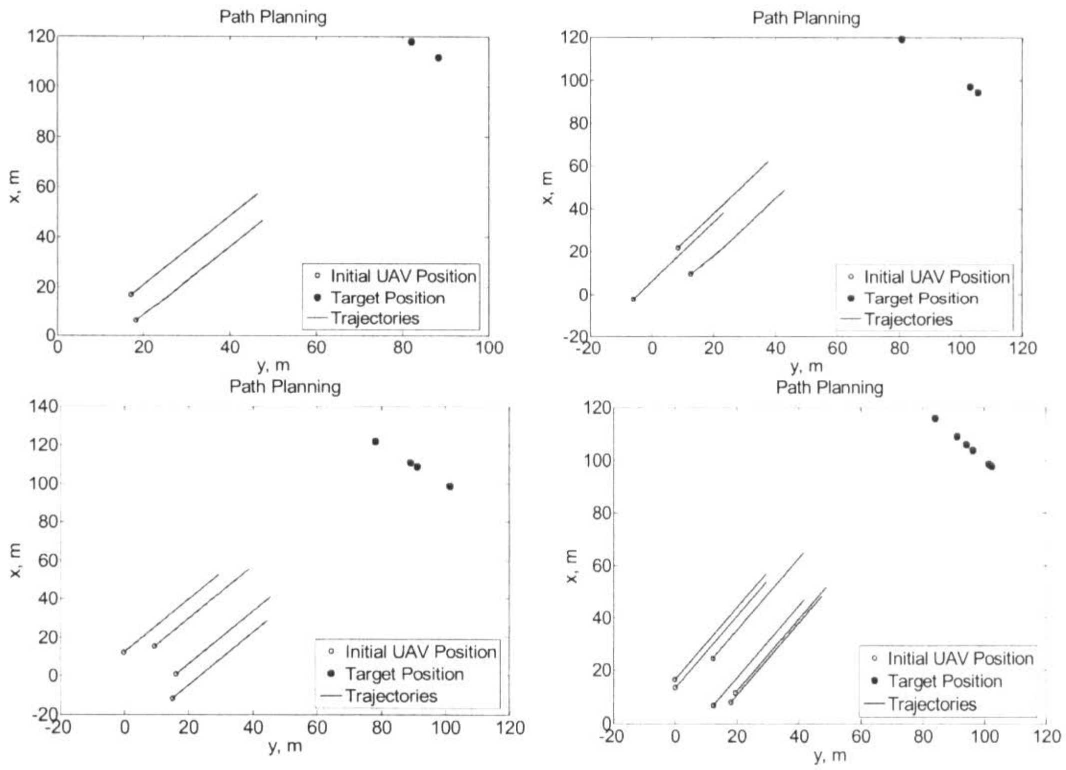


Fig 23. Path planning simulation results for different number of UAVs (during finite horizon time)

As it is shown the UAVs won't reach their assigned target in *T=5* Sec; because, the maximum velocity is limited to *10 m/Sec*. To reach the targets we have to repeat the RHC based path planner till arriving the target position. In the RHC based path planner the trajectory is optimized for a finite horizon time *T* and the first portion of this trajectory will be used during the execution time δ until the next update be available.

A path planning is done for *T=10 Sec* and $\delta = 3\ Sec$, until visiting the targets by UAVs; again B-Spline basis functions of degree 3 are used to generate the paths; the breakpoints are selected as *{0, T}*. For the potential function the following matrix penalties are used:

$$\begin{aligned} K &= 10 \times Ones(N,N) \\ a = b &= 0.1 \times Ones(N,N) \end{aligned} \tag{50}$$

Where *Ones(N,N)* is an *N*-by-*N* matrix that all its elements are one and *N* is the number of UAVs.

Simulation results are shown in figure 24 for two UAVs. In this case for continuity between generated paths, a second order continuity (position, velocity and acceleration) is imposed to the optimization problem as the linear equalities.

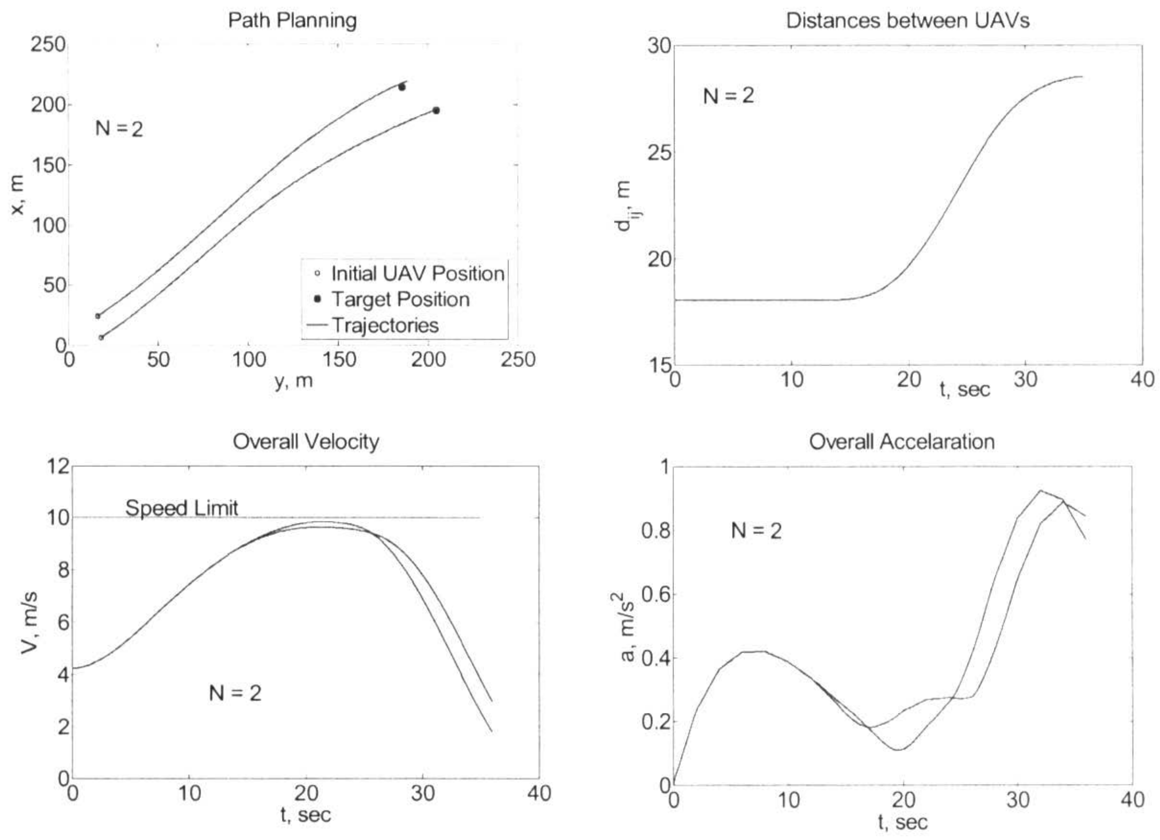


Fig 24. Path planning simulation results by RHC for two UAVs

From the above figure we can see that the UAVs visit their target finally. It is also seen from the above figure that the UAVs keep the appropriate distance and they won't collide each other; hence, the potential functions are suitable tools to manage the distance between neighbour UAVs.

Furthermore, the time history of velocity in figure 24 shows the overall velocity of the UAVs; as it is obvious the maximum allowable velocity (10 m/s) won't be exceeded. The speed limitations are linearized by approximating the circle to polygon. This means this method is very reliable for applying the velocity constraint. The overall acceleration is also shown in figure 24 for both UAVs. For more detail the time history of velocity and acceleration components are depicted in figure 25:

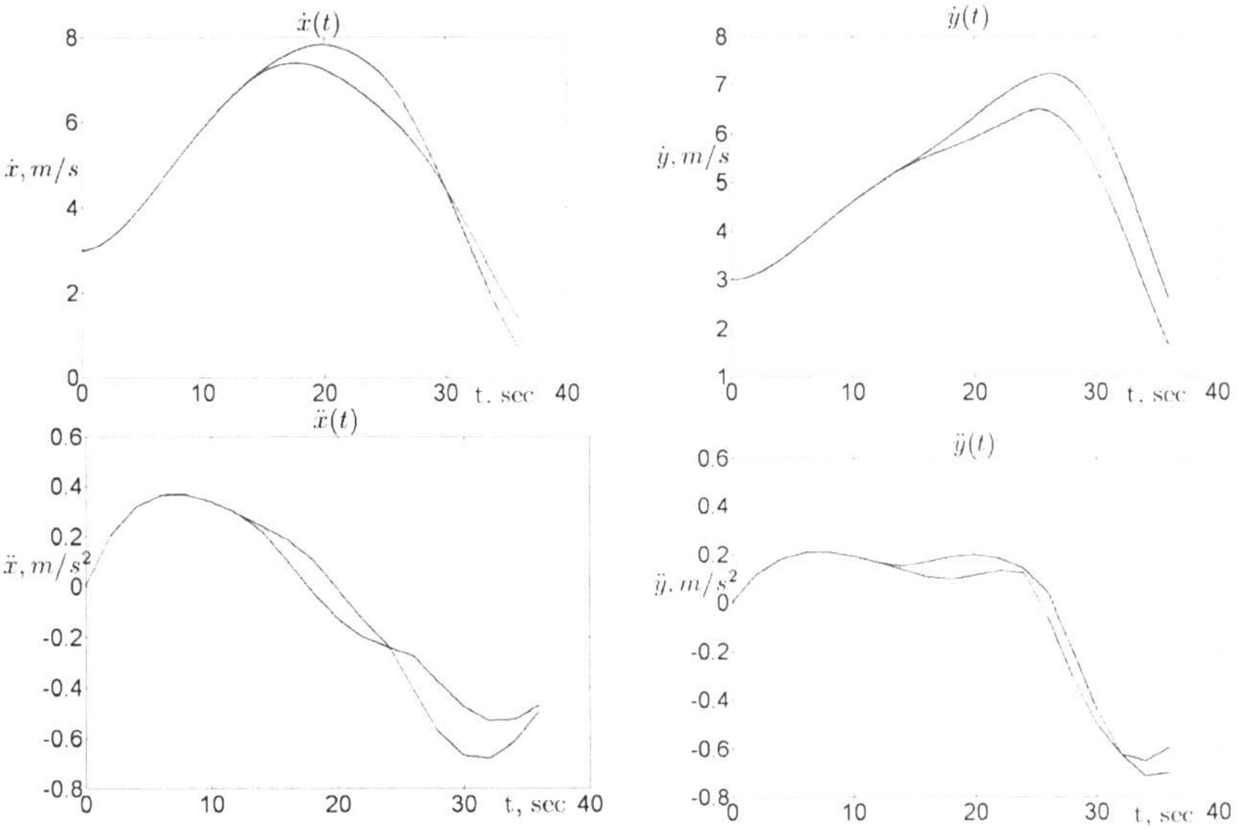


Fig 25. Path planning simulation results by RHC for two UAVs

The simulation results for more number of UAVs are done and the results are depicted in figures 26 and 27. In these cases also it is seen that the proposed receding horizon optimization based path planner generates the paths properly; the UAVs don't collide each other and the maximum velocity isn't exceeded.

Moreover, in all cases, it is seen that the overall computation time is considerably smaller than the simulation time; the maximum t_c / t_r won't exceed 0.5 though the path planning is done for a quite large horizon *T=10 Sec* (t_c is the computation time and t_r is the simulation time or the amount of time that the mission will take in the real time implementation).

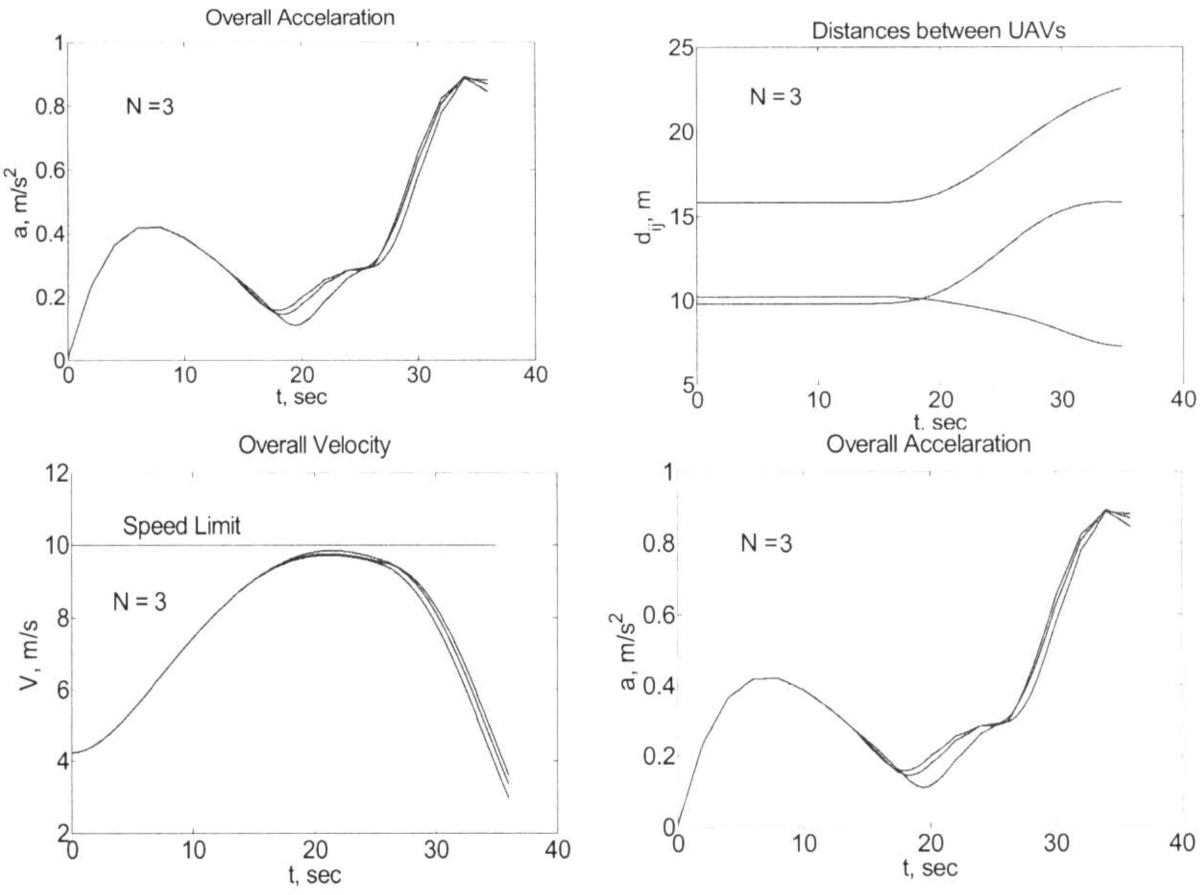


Fig 26. Path planning simulation results by RHC for three UAVs

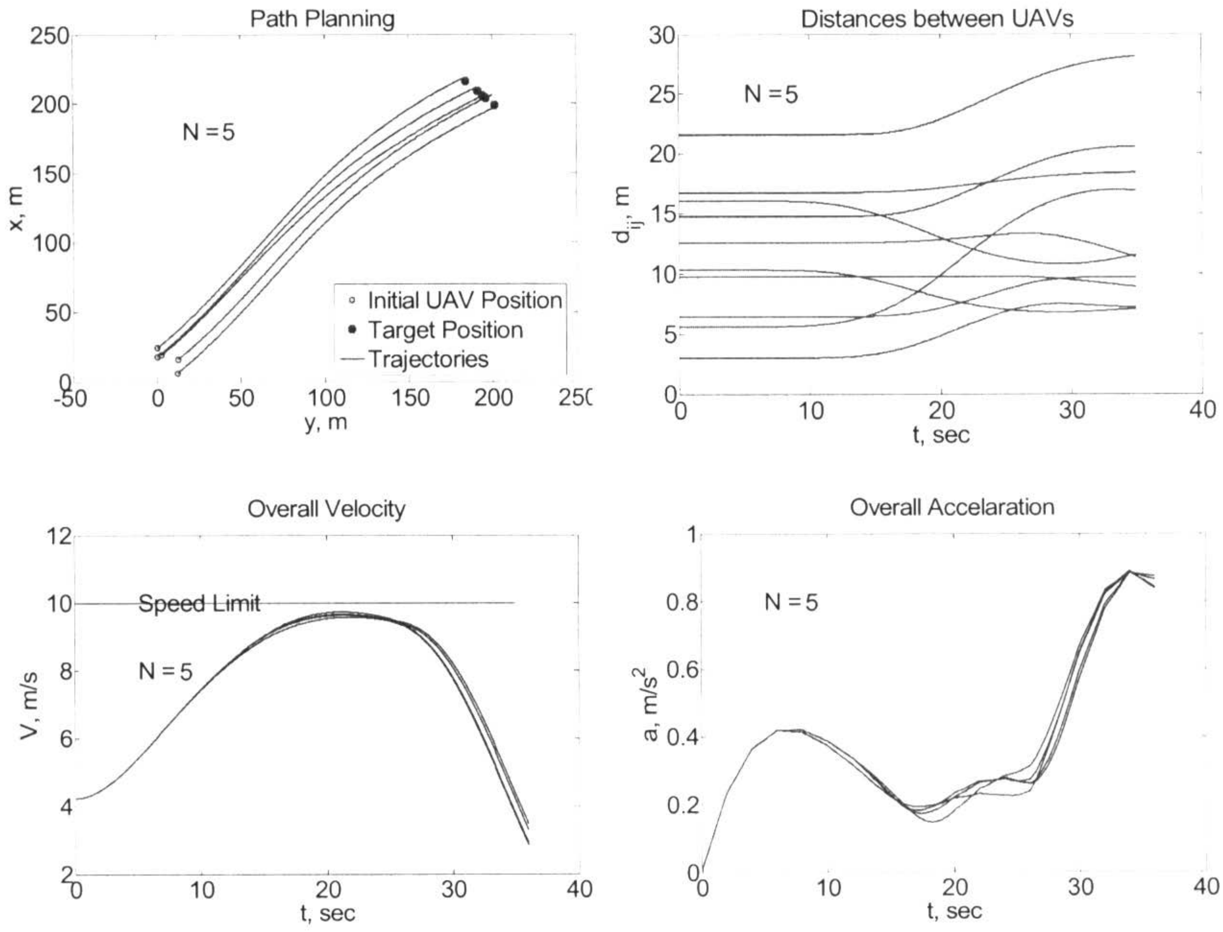


Fig 27. Path planning simulation results by RHC for five UAVs

4.5 Conclusion and Future Work

4.5.1 Conclusion

In this report the fully decentralized control of multi-UAVs has been studied; the problem is divided into three main parts: inner loop controller design, task assignment and path planning. The 3DOF helicopter model is used as the UAV model; also, the receding horizon control (RHC) is used to design an inner loop controller for the 3DOF helicopter model in the stability coordinate. Using B-Spline as the basis function for optimization problem, simulation results show that RHC can generate a feasible control action and it can control the nonlinear and under-actuated helicopter dynamics properly.

For the task assignment and path planning parts, the main objective is to use a method which needs minimum communication among UAVs. The advantages of optimization based methods have been used to achieve this objective; the key idea is that a cost function is given to all UAVs and based on the minimization of this cost function the UAVs make their decisions regarding the task assignment and path planning. Since the global minimum of this cost function is unique for all UAVs, the UAVs will make the same decision.

An optimization based task assignment unit is designed to allocate the targets to each UAV. After that, a receding horizon path planner is designed in a decentralized fashion; simulation results for both task assignment and path planning units show that, the receding horizon optimization based method can successfully control the fleet of UAVs for performing the cooperative missions. The only information that UAVs need is the initial position of other UAVs.

4.5.2 Future work

For the future works, it is proposed to work on the following scenarios; the first one is proposed in section 4.2.1; we can add some obstacles to this scenario. This mission scenario is schematically shown in figure 28:

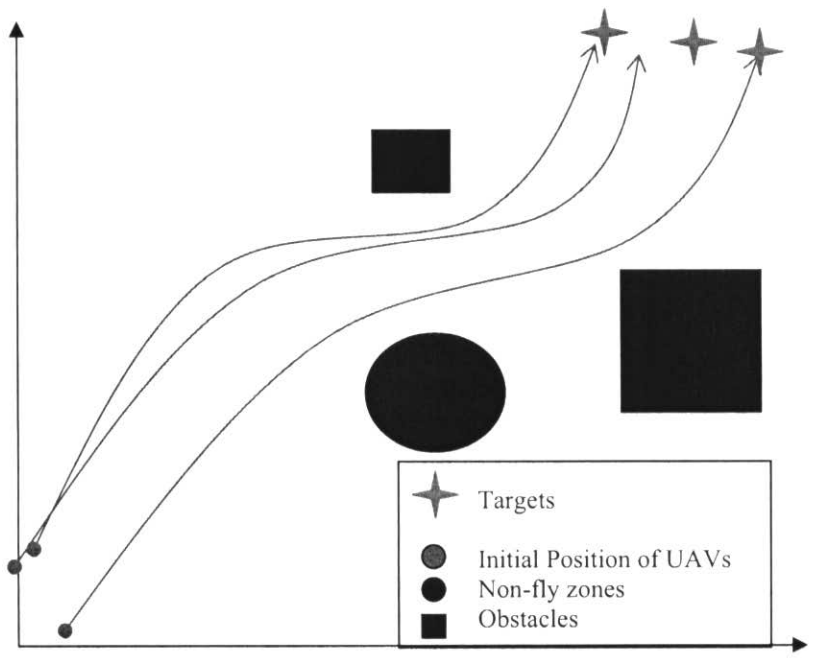


Fig.28. Proposed mission Scenario for future works

N UAVs start from some initial positions (blue circles) and have to visit the targets (red crosses), without colliding to each other and obstacles (black squares) and they shouldn't fly in non-fly zones (black circles) where there is a probability for existence of enemy radars.

In this report, the "fully decentralized path planner" is discussed; for the future it is proposed to work on the "overlapping decentralized path planner" so that the communicated information can be used efficiently to improve the performance of path planning. Also, it is desired to limit the communication to only neighbour UAVs and not all UAVs. In fact, the communication is done locally among the neighbour UAVs and not all UAVs.

Also, the certainty level of information can be reduced in order to study and increase the robustness of the decentralized method. Since in the real-world applications, environmental information often is known with a limited level of certainty and also the environment is dynamic, this can make the scenario more realistic.

4.6 Appendix A: Flat Outputs

The dimension of the optimal control problem is reduced to a lower dimension that easing the formulation and reducing the amount of required computations for optimization process by means of so-called flat outputs. For applying the RHC method and solving the optimal control problem the flat output advantages has been utilized. System (26) is called a flat system if there exist outputs z [11]:

$$Z=g(x,u) \tag{51}$$

Such that the states and the control signal can be recovered from z and its derivatives; that is, all states and control inputs can be presented as a function of flat outputs and their derivatives:

$$(x,u)=h(z,\dot{z},....,z^{(r)}) \tag{52}$$

Equation (52) is only required to hold locally [11]. For a system being flat, it is required that all system behaviours including states and control inputs can be recovered from a finite number of flat outputs and their derivatives and without integration of the flat outputs; hence, h has to be a smooth function. The interested readers are referred to [11] for more details. If the dimension of z is smaller than that of x and u, the optimization problem is mapped to a lower dimension.

4.7 References

[1] Matthew C. Lechliter, "Decentralized Control for UAV Path Planning and Task Allocation", MS Thesis, Department of Mechanical and Aerospace Engineering, West Virginia University, Morgantown, WV, 2004.

[2] Mehdi Alighanbari and Jonathan P. How, "Decentralized Task Assignment for Unmanned Aerial Vehicles", Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005, Seville, Spain, December 12-15, 2005.

[3] J. Bellingham, M. Tillerson, A. Richards and J. P. How, "Multi-Task Allocation and Trajectory Design for Cooperating UAVs" in Cooperative Control: Models, Applications and Algorithms at the Conference on Coordination, Control and Optimization, November 2001, pp. 1-19.

[4] Johan L¨ofberg, "Nonlinear Receding Horizon Control: Stability without Stability Constraints", Internal Report No.: LiTH-ISY-R-2356, Division of Automatic Control, Department of Electrical Engineering, Link¨opings universitet, SE-581 83 Link¨oping, Sweden, 2001.

[5] Roskam. J., "Airplane Flight Dynamics and Automatic Flight Control - Part I, Roskam Publishing Incorp., 1979.

[6] V. Gavrilets, B. Mettler and E. Feron "Dynamic Model for a Miniature Aerobatic Helicopter", MIT Internal Report.

[7] William Dunbar, "Distributed Receding Horizon Control of Multi agent Systems", PhD thesis, Control and Dynamical Systems, California Institute of Technology, April, 2004.

[8] Yan Liao, Yan Jin, Ali A. Minai and Marios M. Polycarpou "Information Sharing in Cooperative Unmanned Aerial Vehicle Teams" Proceedings of the 44th IEEE Conference on

Decision and Control, and the European Control Conference 2005 Seville, Spain, December 12-15, 2005.

[9] M. B. Milam, R. Franz, J. E. Hauser and R. M. Murray, "Receding horizon control of a vectored thrust flight experiment", submitted to IEE Proceedings on Control Theory and Applications.

[10] Carl de Boor. "A Practical Guide to Splines", Springer-Verlag, New York, 2001.

[11] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems", PhD Thesis, California Institute of Technology, Pasadena, CA, 2003.

[12] H. A. Izadi, B. W. Gordon, "Robust Quasi-Infinite Receding Horizon Control Approach for Vortex Break down Augmented Roll Dynamics of Delta Wing Aircraft", internal report submitted to DRDC, October, 2005.

[13] Alvaro E. Gil, Kevin M. Passino and Jose B. Cruz, Jr. "Stable Cooperative Surveillance", Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005 Seville, Spain, December 12-15, 2005.

[14] Waseem Ahmed Kamal, Da-Wei Gu and Ian Postlethwaite, "Real Time Trajectory Planning for UAVs Using MILP" Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005 Seville, Spain, December 12-15, 2005.

[15] Arthur Richards and Jonathan How, "Decentralized Model Predictive Control of Cooperating UAVs", 43rd IEEE Conference on Decision and Control, December 14-17, 2004, Atlantis, Paradise Island, Bahamas.

[16] T. Schouwenaars, J. P. How, and E. Feron, "Receding Horizon Path Planning with Implicit Safety Guarantees" Proceedings of the IEEE American Control Conference, 2004, pp. 5576 - 5581.

[17] T. Schouwenaars, B. de Moor, E. Feron, and J. P. How, “"Mixed Integer Programming for Multi-vehicle Path Planning”, Proceedings of the European Control Conference, European Union Control Association, Porto, Portugal, September, 2001, pp. 2603-2608.

[18] J. Bellingham, A. Richards, and J. P. How, “Receding Horizon Control of autonomous Aerial Vehicles” Proceedings of the IEEE American Control Conference, May 2002, pp. 3741-3746.

[19] H.L. Hagenaars, J. Imura and H. Nijmeijer “Approximate Continuous-time Optimal Control in Obstacle Avoidance by Time/Space Discretization of Non-convex State Constraints”, IEEE Conference on Control Applications, 2004, pp. 878-883.

[20] Bortoff, S. A. “Path-Planning for Unmanned Air Vehicles” Air Vehicles doctorate, right-Patterson Air Force Base, Ohio, August 1999.

[21] Kavraki, et al., “Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces,” *IEEE Transactions on Robotics and Automation*, Vol. 12, No. 4, 1996, pp. 566-580.

[22] Overmars, M. and Svestka, P., “A Probabilistic Learning Approach to Motion Planning,” *Algorithmic Foundations of Robotics, the 1994 Workshop on Algorithmic Foundations of Robotics*, edt. Goldberg, Halperin, Latombe, and Wilson, 1995.

[23] Song, G., Miller, S., and Amato, N., “Customizing PRM Roadmaps at Query Time,” *Proceedings of IEEE International Conference on Robotics and Automation*, 2001.

[24] Yoshiaki Kuwata and Jonathan P. How “Stable Trajectory Design for Highly Constrained Environments using Receding Horizon Control”, Proceeding of the 2004 American Control Conference Boston, Massachusetts June 30 - July 2, 200

CHAPTER 5: REAL-TIME SCHEDULING OF MULTIPLE UNCERTAIN RECEDING HORIZON CONTROL SYSTEMS

(Task # 7, 8, 9)

Ali Azimi
PhD Student &
Research Assistant

Behnood Gholami
MASc Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory

Department of Mechanical and Industrial Engineering

Concordia University

Montreal, Quebec, Canada H3G 1M8

March 2006

Abstract

This report documents the work related to tasks #7, 8 and 9 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". This report describes the design of a real time scheduling algorithm for multiple uncertain receding horizon control systems. Experimental verification has been performed using a set of 3DOF miniature under actuated hovercrafts. This demonstrates the performance and feasibility of the proposed method. The performance of the method is also demonstrated by simulating it in Matlab. Furthermore, an RHC Object Oriented Library (RHCOOL) developed in the CIS lab is described in detail. This library greatly simplifies the implementation of scheduled RHC hard real-time simulations and experimental implementations for multiple UAV systems.

5.1 Introduction

Scheduling of RHC computational and communication is very important for decentralized RHC control since it allows optimization of resources to achieve maximum performance in the presence of uncertainty, disturbances, and delays. The tasks addressed in this report related to the project entitled "Real-time Scheduling of Multiple Uncertain Receding Horizon Control Systems" are as follows:

Task 7: Design a baseline run-time scheduling algorithm that provides connections with constrained optimization performance.

Task 8: Demonstrate feasibility, performance through Matlab and via single and two processor computing.

Task 9: Document results, theory and practice

For these tasks we designed a baseline run-time scheduling algorithm that provides connections with constrained optimization performance (task #7). We extended and improved the results of the CIS lab on scheduling of multiple RHC computation problems. In this work, static scheduling algorithms for a single processor that indirectly optimize robust RHC performance are introduced to achieve good robust performance. This is achieved through minimization of the difference between the nominal and actual models subject to scheduling constraints.

Feasibility and performance of the proposed method was demonstrated through Matlab and also by applying the method to some sets of 3DOF miniature under actuated hovercrafts. The latter was done first, by using two hovercrafts with both one and two processors and second by using four hovercrafts with 2 processors (task #8).
In order to be able to apply the RHC method to the miniature hovercrafts, and then apply scheduling algorithm on them, first a model of those hovercrafts should be discovered and the parameters of that model have to be identified. These tasks have been done and the results are shown in this report. The following sections are included in this report.

In section two, real-time scheduling of multiple uncertain RHC systems has been developed. In this section, a new scheduling approach is proposed that considers the effect of modeling uncertainty for multiple continuous time RHC systems. This is accomplished by combining a scheduling approach with results from continuous time nonlinear systems theory. Assuming the prediction horizon is a known constant and using a single computing resource, a new technique is proposed for determining the execution horizon for multiple RHC systems. The problem of determining the execution horizon for each subsystem while optimizing the performance is cast into a constrained optimization problem.

In section three, application to 3DOF under-actuated miniature hovercraft problem has been investigated. Since RHC approach is a model-based algorithm, modeling and identification of the apparatus has been done. A Receding Horizon Control Object Oriented Library (RHCOOL) has been developed in C++. This library has the benefit of applying different RHC approaches in a hard real-time environment to multi UAVs, hovercrafts, and etc. Finally, using the RHCOOL, the real-time scheduling of hovercraft has been investigated based on the designed scheduling method.

In section four, experimental results have been reported for miniature hovercrafts. The results are from implementing RHC approach to single hovercraft as well as scheduling results of one processor with two hovercrafts.

Conclusions and future work are presented in section five and references are in section six.

Section 2 and section 3.2 of this report are describing Task #7 and the other sections of the report are describing Task #8 of the first proposal.

5.2 Real-time Scheduling of Multiple Uncertain RHC Systems

This section describes the scheduling approach recently developed at the CIS lab [1]. Receding Horizon Control (RHC), also known as Model Predictive Control (MPC), was first introduced in the process control community. It has attracted the attention of many researchers due to its ability to handle constraints on the states and inputs in multi-variable control problems [2]. RHC is essentially a repeated on-line solution to a finite-horizon open-loop optimal control problem. Based on the currently available state values, an optimal control problem is solved for a period of time called the *prediction horizon*. The first part of the computed optimal control input is applied to the plant in a period of time called the *execution horizon* until the next sample of the state becomes available, where again the same procedure is repeated. Therefore, the execution horizon can be considered as the closed loop sampling period[1] for the RHC system and the two terms used interchangeably.

In the past, the computational cost of the repeated on-line optimization problems associated with RHC limited its applicability to relatively slow dynamic systems such as chemical processes. However, with recent advances in computing performance and distributed computation, RHC can now be applied to plants with fast dynamics including aerospace systems. In [5], an efficient direct method for solving optimal control problems has been proposed based on the properties of *flat outputs*. The method proposed in [5] was applied successfully to a vector thrust flight experiment in [6]. A number of methods have been proposed to provide closed-loop stability for RHC systems where a terminal cost or terminal constraint is introduced into the optimization problem. For a detailed survey the reader is referred to [3]. Methods to improve the performance of RHC in the presence of computational delays have also been recently developed [21].

Application of RHC to control problems with multiple subsystems such as multiple Unmanned Ariel Vehicle (UAV) systems can be addressed by applying RHC to the individual subsystems, which may contain more than a single CPU. This results in multiple RHC processes that must be scheduled in an appropriate manner to achieve optimal performance in the presence of computing resource limitations on-board each vehicle and across networked computing devices. Systematic methods for scheduling RHC systems are typically not discussed in the literature, despite the fact that 1) computational delays resulting from scheduling can dramatically affect stability and performance, and 2) significant decrease in computing time could potentially be achieved. The authors in [7] considered the scheduling of multiple discrete time decoupled linear RHC systems with a single computational resource. For this case a dynamic priority assignment policy is proposed that gives higher scheduling priority to RHC subsystems with larger current cost function values. Premature termination of the optimization processes is also employed to maximize nominal performance in the

[1] Sometimes the execution horizon maybe considered bigger than one sampling period. For example in this report when two apparatus are scheduled using one computer, the sampling period is less than the execution horizon of each subsystem.

presence of computational delays. These results provide a new RHC scheduling approach for increased nominal performance of discrete time systems without explicitly accounting for uncertainty in the subsystems.

In this section, a new scheduling approach is proposed that considers the effect of modeling uncertainty for multiple continuous time RHC systems. This is accomplished by combining a scheduling approach with results from continuous time nonlinear systems theory. Assuming the prediction horizon is a known constant and using a single computing resource, a new technique is proposed for determining the execution horizon for multiple RHC systems[2]. A rate monotonic priority assignment [8] is employed due to the fact that it is an optimal policy in the sense that the task set is not schedulable using any static priority assignment method if it is not schedulable using the rate monotonic method [9], [10]. The problem of determining the execution horizon for each subsystem while optimizing the performance is cast into a constrained optimization problem. The schedulability condition is represented by a constraint and the robust performance is formulated as the objective function. This section can be regarded as the first systematic approach for scheduling and execution horizon selection of multiple uncertain RHC systems subject to computing resource limitations. The new method is applied for a two subsystem example problem demonstrating the effectiveness of the approach.

5.2.1 Receding Horizon Control of Nonlinear Systems

The class of systems considered in this report is described by the set of equations

$$\dot{x} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{1}$$

where $\boldsymbol{x}(t) \in \Re^n$ is the state of the system and $\boldsymbol{u}(t) \in \Re^m$ is the input vector satisfying the constraints

$$\boldsymbol{u}(t) \in U \qquad \forall t \geq 0 \tag{2}$$

where U is the allowable set of inputs. Furthermore, it is assumed that (A1-A3) in [11] are also satisfied; that is, f is twice differentiable, U is compact and convex, and system (1) has a unique solution for a given initial condition. Receding horizon control is the repeated solution of the following problem.

Problem 1: Find

[2] In applying RHC to a system in real-time, periodic function is defined and the period of that periodic function is equal to the execution horizon. In scheduling of multiple RHC systems, multiple periodic functions need to be defined. The important issue is to define the period of each function which is equal to the execution horizon of each system. After defining the execution horizon of each subsystem, the priority of each task is defined by using Rate Monotonic Priority Assignment.

$$J_T^*(\boldsymbol{x}(t)) = \min_{\boldsymbol{u}(.)} J(\boldsymbol{x}(t), \boldsymbol{u}(.), T) \tag{3}$$

with

$$J(\boldsymbol{x}(t), \boldsymbol{u}(.), T) = \int_t^{t+T} \left(\left\| \boldsymbol{x}(\tau; \boldsymbol{x}(t)) \right\|_{\mathbf{Q}}^2 + \left\| \boldsymbol{u}(\tau) \right\|_{\mathbf{R}}^2 \right) d\tau + \left\| \boldsymbol{x}(t+T; \boldsymbol{x}(t)) \right\|_P^2 \tag{4}$$

subject to

$$\left. \begin{aligned} \dot{\boldsymbol{x}}(s) &= \boldsymbol{f}(\boldsymbol{x}(s), \boldsymbol{u}(s)) \\ \boldsymbol{u}(s) &\in U \end{aligned} \right\} \quad s \in [t, t+T] \tag{5}$$

$\mathbf{Q} \in \Re^{n \times n}$ and $\mathbf{R} \in \Re^{m \times m}$ denote positive-definite, symmetric weighting matrices, T is a finite prediction horizon and $\boldsymbol{x}(t; \boldsymbol{x}_0)$ denotes the trajectory of the system (1) driven by $u(t)$ starting from the initial condition x_0. Furthermore, the weighted norms in (4) are defined as $\|\boldsymbol{x}\|_{\mathbf{Q}}^2 = \boldsymbol{x}^T \mathbf{Q} \boldsymbol{x}$.

Let δ denote the RHC execution horizon, where δ lies in the $(0,T]$ interval. The closed-loop system is described by

$$\begin{aligned} &\dot{\boldsymbol{x}}(\tau) = \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}^*(\tau)) \\ &\boldsymbol{u}^*(\tau) = \boldsymbol{u}^*(\tau; \boldsymbol{x}(t_k)) \qquad \tau \in [t_k, t_k + \delta), \quad 0 < \delta \le T \end{aligned} \tag{6}$$

where $\mathbf{u}^*(\tau; \mathbf{x}(t_k))$, $\tau \in [t_k, t_k + T]$ is the optimal control with initial condition $\boldsymbol{x}(t_k)$, t_k being the start time of the optimization process and the instant at which states are sampled. The subscript k in t_k denotes the kth sampling time and therefore $t_{k+1} = t_k + \delta$.

The optimization problem described above can be solved numerically online using a number of techniques [6]. Details of the solution procedure for this section are described in section 2.3. Numerous methods have been suggested to guarantee the stability of the closed-loop system by requiring a terminal constraint or a special way to select the terminal cost [3], [11]. Note that the RHC method can be formulated in discrete time or continuous time as is the case for this report. Discretization is done by zero order hold sampling of the continuous time.

5.2.2 Real-time Scheduling of Multiple Uncertain RHC Systems

5.2.2.1 Real-time Computation of RHC Systems

The open-loop RHC optimal control problem can be solved in a periodic manner with a period of δ. The sampling instant is denoted by t_k and the next sample time is given by $t_{k+1} = t_k + \delta$. A number of parameters, which are important for practical implementation of the RHC algorithm, are defined below (refer to Figure):

- *Computation start time, t_{sc}*: The time at which the optimization process is started.
- *Computation end time, t_{ec}*: The time at which the optimization process is finished.
- *Computation time, δ_c*: The duration of the optimization. $\delta_c = t_{ec} - t_{sc}$
- *Actuation time, t_a*: The time at which the generated control input is applied to the system. This input might be recently computed or generated in advance depending on the implementation method used.
- *Actuation latency, l_a*: The delay involved in the actuation of the plant after the sampling of the states.

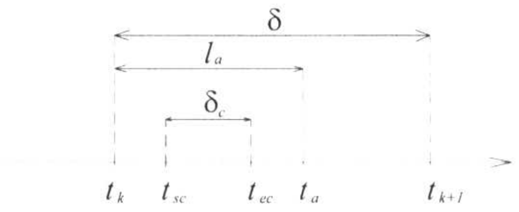


Figure 1- Schematic diagram for RHC parameters

In theoretical developments of RHC, the computation time is typically assumed to be zero [11], [12], *i.e.* $\delta_c = 0$. However, in practical real-time control systems this is not a valid assumption, especially for systems with fast dynamics. In such systems (*e.g.* see [6]), unlike slower traditional process control problems, a small execution horizon (sampling period) is needed for satisfactory performance of the system. This leads to the situation where the computation time is not negligible compared to the execution horizon, and a zero computation time assumption is invalid. There are two methods available to apply RHC to practical systems when the computation time is not negligible. These are referred to as *Retarded Actuation* and *On-the-Fly Computation.*

In the *Retarded Actuation* approach, for the control system to have sufficient time for generating the optimal trajectories, the optimization problem is solved one sampling time in advance ([6]). The control signal can also equivalently be actuated one sampling time later. The implicit assumption of the method is that the computation time is less than the sampling period δ. The schematic diagram for this method is shown in Figure 2. There are two variations of this method:

(i) The control signal generation for the interval $[t_{k+1}, t_{k+2}]$ starts at t_k, t_k being the k^{th} sampling time and $t_{k+1} = t_k + \delta$. The open-loop optimal control problem is solved using $x(t_k)$ as the initial condition. The part of the optimal input u^* corresponding to the time interval $[\delta, 2\delta]$ is applied to the system.

(ii) In this variation, instead of using $x(t_k)$, $x(t_{k+1})$ is predicted using the nominal model. The predicted initial conditions are then used for solving the open-loop optimal control problem. Contrary to the previous method, the part of the optimal input u^* corresponding to the time interval $[0,\delta]$ is applied to the system.

The following properties hold, in terms of the parameters defined earlier, for the two methods described above

$$\begin{aligned} t_{sc} &= t_k \\ t_{ec} &\leq t_{k+1} \\ t_a &= t_{k+1} \\ \delta_c &\leq \delta \\ l_a &= \delta \end{aligned} \tag{7}$$

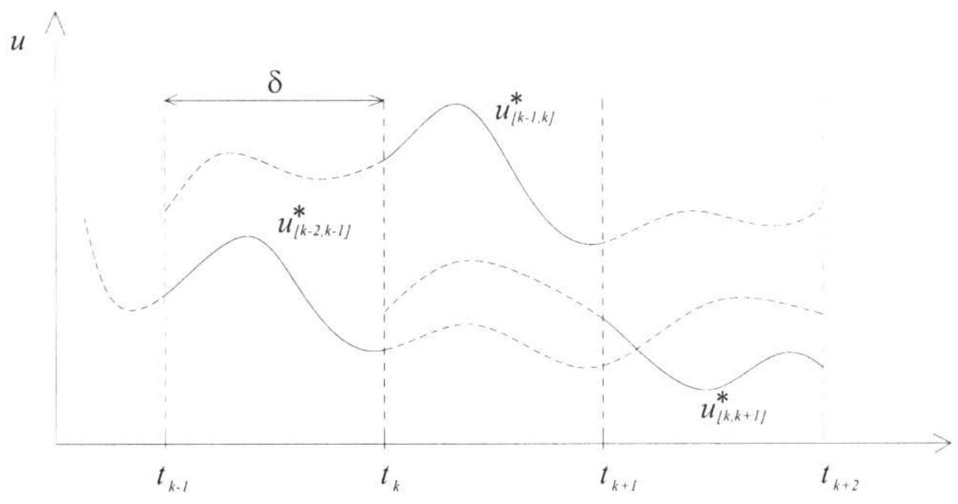


Figure 2- Schematic diagram showing the *Retarded Actuation* method. The implemented input is shown by a solid line. The subscript refers to the interval in which the input profile has been generated.

For the *On-the-Fly Computation* approach, there is no prediction or delay in the actuation of the input signal (*i.e.* l_a=0). After sampling the states, optimization starts and as soon as the solution is found the control signal is applied. While the optimization is running, the optimal control input from the previous solution is being implemented. Since the *prediction horizon* is normally much longer than the *execution horizon*, the method can continue implementing the previously computed control input until the new input is generated. Therefore, the actual implemented input profile is composed of two parts: The continuation of the optimal input available from the previous sampling and the newly generated optimal input. This is shown in the schematic diagram depicted in Figure 3.

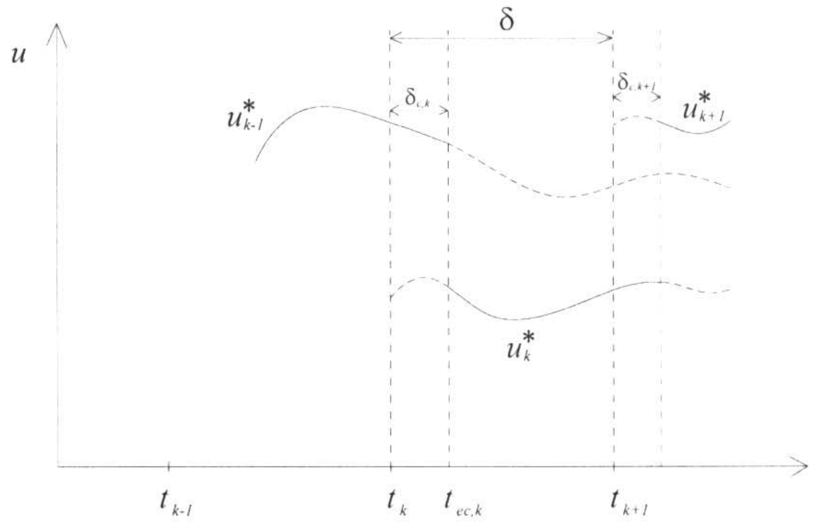


Figure 3- Control signal when the *On-the-Fly Computation* method is used

The following properties hold for this method

$$\begin{aligned} & t_{sc} = t_k \\ & t_{ec} \leq t_{k+1} \\ & t_a < t_{k+1} \\ & \delta_c \leq \delta \\ & l_a = t_{ec} - t_k \end{aligned} \tag{8}$$

This report employs the *Retarded Actuation* method since it allows more deterministic behaviour than the *On-the-Fly Computation* approach used in [7]. This is helpful when analyzing the effect of computational delays using nonlinear systems theory and combining the approach with existing scheduling theory. The *On-the-Fly Computation* approach potentially provides better nominal performance, but it is more difficult to analyze in a deterministic framework. Furthermore, new methods have been proposed by the authors recently to improve the performance of the *Retarded Actuation* method using neighbouring extremal paths theory [21].

As discussed in [7], the delay introduced in actuating the plant due to the computation time can degrade the performance of the system considerably. For this reason it is desirable to schedule the computation to improve the performance of RHC systems.

5.2.2.2 Scheduling Approach for Multiple Uncertain RHC Systems

In this section a systematic scheduling approach is proposed to determine the execution horizon for each individual subsystem when the *Retarded Actuation* method (type ii, section 2.2.1) and rate monotonic priority assignment are employed. It is assumed that the optimization associated with the RHC problems converges in a bounded computational time so that the computation can be scheduled in a hard real-time environment. It will be shown that in order to maximize the overall performance while the *schedulability* property holds, a constrained optimization problem must be solved. Schedulability implies a guarantee that the optimal trajectory generation for each subsystem using a single computer is finished before the deadline.

Trajectory generation for multiple systems using a single computing resource can be considered a resource allocation problem described in computer science literature. The following definitions from this area will thus be reviewed before addressing the main problem of this report [30].

Definition 1. The time required to complete a task (*e.g.* solving the optimization problem in our case) is called the *execution time* (see Figure 4).

Remark 1. The execution time should not be confused with the execution horizon defined for RHC systems. Execution time is the same as the computation time, $\delta_{c,I}$, defined in Section 5.2.2 when the trajectory generation problem is considered.

Definition 2. Consider a task i that should be executed every p_i seconds. This task is defined as a *periodic* task with a *period* of p_i (see Figure 4).

The period can be fixed or time-varying. In this report only tasks with fixed periods are considered.

Definition 3. The CPU utilization factor, μ, for a series of n tasks is defined as follows

$$\mu = \sum_{i=1}^{n} \frac{\delta_{c,i}}{p_i} \leq 1 \tag{9}$$

where $\delta_{c,i}$ and p_i represent the execution time and period of task i. For a single computing resource μ must be less than one.

Definition 4. The execution of task i should be finished before a certain pre-determined time. This pre-determined time is called the *deadline*[3] of task i (see Figure 4).

Definition 5. A series of tasks are called *schedulable* if all tasks can be executed while no deadline has been missed.

[3] The *deadline* is equal to the *period* for the periodic tasks. In addition, defining periodic tasks is a common way for real-time operation of control tasks [35].

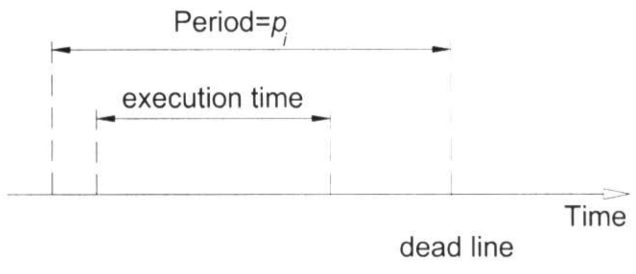


Figure 4- Schematic diagram for a periodic task

Definition 6. A task is called *preemptive* if its execution can be interrupted by the execution of a higher priority task.

Definition 7 (Rate Monotonic Priority Assignment) [8]. Given a set of preemptive tasks with static priorities, the higher priority is assigned to the task with the smaller period.

Remark 2. The following inequality is a sufficient condition to guarantee schedulability for a set of n tasks [8]

$$\mu = \sum_{i=1}^{n} \frac{\delta_{c,i}}{p_i} \leq n(2^{\frac{1}{n}} - 1) \tag{10}$$

The above is a sufficient but not necessary condition. In other words, a task set that does not satisfy the above inequality can still meet all deadlines using Rate Monotonic Scheduling (RMS) [39]. Rate Monotonic Scheduling is optimal under the above condition (inequality). RMS has a nice advantage. Also, in many situations, up to 90% utilization is possible with RMS [39]. In order to get more information about the different schedulability conditions see [40] and [41].

The rate monotonic method will be employed in this report for static priority assignment in scheduling a set of tasks. This approach is selected, among numerous other methods, due to the fact that rate monotonic priority assignment is an optimal policy. The task set is not schedulable using any static priority assignment method if it is not schedulable using the rate monotonic method as explained in [9], [10].

The proposed scheduling algorithm can now be expressed in the following proposition.

Proposition 1. Consider the problem of controlling n decoupled uncertain nonlinear systems using the RHC scheme. The following equations serve as the nominal model for describing the decoupled subsystems

$$\begin{aligned} \dot{\boldsymbol{x}}_1 &= \boldsymbol{f}_1(\boldsymbol{x}_1(t), \boldsymbol{u}_1(t)) \\ &\vdots \\ \dot{\boldsymbol{x}}_n &= \boldsymbol{f}_n(\boldsymbol{x}_n(t), \boldsymbol{u}_n(t)) \end{aligned} \tag{11}$$

The actual subsystem are described by

$$\begin{aligned}
\dot{y}_1 &= f_1(y_1(t), u_1(t)) + g_1(t, y_1, u_1) \\
&\vdots \\
\dot{y}_n &= f_n(y_n(t), u_n(t)) + g_n(t, y_n, u_n)
\end{aligned} \tag{12}$$

In equation (12), $g_i(t,y,u)$, $i=1,2,...,n$ accounts for disturbances and modeling uncertainty and its norm is upper bounded by b_i. The function f_i in equations (11) and (12) is *Lipschitz* continuous in the domain of operation of the system with a *Lipschitz* constant represented by L_i.

It is useful to note that the application of proposed scheduling method to multiple UAVs, is to control some UAVs with limited computational resource; for example controlling 2 UAVs using only one computer. In this case the nominal model used in controller (to predict the states and inputs over the prediction horizon) can be considered as

$$\begin{aligned}
\dot{\mathbf{x}}_1 &= \mathbf{f}_1(\mathbf{x}_1(t), \mathbf{u}_1(t)) \\
\dot{\mathbf{x}}_2 &= \mathbf{f}_2(\mathbf{x}_2(t), \mathbf{u}_2(t))
\end{aligned} \tag{13}$$

while the actual UAVs have different models. The source of this difference between two models can be, for example, because of the friction or the other model uncertainty which may exist. Here the actual model is represented by

$$\begin{aligned}
\dot{\mathbf{y}}_1 &= \mathbf{f}_1(\mathbf{y}_1(t), \mathbf{u}_1(t)) + \mathbf{g}_1(t, \mathbf{y}_1, \mathbf{u}_1) \\
\dot{\mathbf{y}}_2 &= \mathbf{f}_2(\mathbf{y}_2(t), \mathbf{u}_2(t)) + \mathbf{g}_2(t, \mathbf{y}_2, \mathbf{u}_2)
\end{aligned} \tag{14}$$

There are two differences between the actual and nominal models:

1- The state variables are different, because in nominal model they are predicted based on the nominal model and in reality they are affected by the real system. Here $\mathbf{x}_i$ represent the predicted states from nominal model and $\mathbf{y}_i$ represent the actual states.

2- The other difference is the difference between two models because of the uncertainty which discussed before. $\mathbf{g}_i$ represents the foresaid difference.

If there is only one computing resource available for this set of n subsystems, the execution horizon for each subsystem should be chosen so that the set of tasks remain schedulable. This should be carried out considering the fact that the level of uncertainty in each subsystem is different.

The following scheduling approach can be used to minimize the state prediction error of the uncertain subsystems. If the rate monotonic priority assignment method is used, the solution to the following optimization problem minimizes the overall upper bound of the state prediction error on all subsystems, while the set of tasks remain schedulable.

$$\begin{aligned}&\text{minimize} \sum_{i=1}^{n} \alpha_i \frac{b_i}{L_i}(e^{L_i\delta_i}-1)\\ &\text{s.t.} \begin{cases} \sum \dfrac{\delta_{c,i}}{\delta_i} \leq n(2^{\frac{1}{n}}-1) \\ 0<\delta_i \leq \delta_{ub,i} \end{cases}\end{aligned} \tag{15}$$

where α_i and $\delta_{c,i}$ represent the weighting parameter and computation time for task i respectively. $\delta_{ub,i}$ is the upper bound for the execution horizon which is usually some fraction of the prediction horizon. L_i is the Lipschitz constant of the function $\boldsymbol{f}_i$. In the proposed optimization problem the optimization parameters are the execution horizons for each subsystem, δ_i, $i=1,2,\ldots,n$.

Proof. As described in Section 2.2.1 (*Retarded Actuation* method, type ii), the RHC implementation scheme involves prediction of the states at the next sampling time and using them as initial conditions to solve the open-loop optimal control problem in advance. The prediction error for subsystem i is described by $\left\|\boldsymbol{y}_i(t_{k,i}+\delta_i)-\boldsymbol{x}_i(t_{k,i}+\delta_i)\right\|$. The right hand side of the following inequality, which is a direct consequence of the Bellman-Grownwall Lemma, is an upper bound on the prediction error for each subsystem [13],[14]

$$\left\|\boldsymbol{y}_i(t_{k,i}+\delta_i)-\boldsymbol{x}_i(t_{k,i}+\delta_i)\right\| \leq e^{L_i\delta_i}\left\|\boldsymbol{y}_i(t_{k,i})-\boldsymbol{x}_i(t_{k,i})\right\|+\frac{b_i}{L_i}\left(e^{L_i\delta_i}-1\right) \quad i=1,2,\ldots,n \tag{16}$$

where $t_{k,i}$ is the time at which sampling of the states for subsystem i has taken place. Evidently $\boldsymbol{x}(t_{k,i})=\boldsymbol{y}(t_{k,i})$ for $i=1,2,\ldots,n$ since $t_{k,i}$ is the time at which the sampling has taken place and there is no mismatch between the states of the nominal and actual system. Therefore (16) reduces to

$$\left\|\boldsymbol{y}_i(t_{k,i}+\delta_i)-\boldsymbol{x}_i(t_{k,i}+\delta_i)\right\| \leq \frac{b_i}{L_i}\left(e^{L_i\delta_i}-1\right) \quad i=1,2,\ldots,n \tag{17}$$

The right hand side of equation (17) justifies the chosen form of the objective function in (15). Furthermore, for a series of task to be schedulable equation (10) should hold. This together with an upper bound on the desired execution horizon for each subsystem constitutes the constraints of the optimization problem. $\square$

Remark 2. In [15] the author discusses the fact that a larger error in the prediction of the initial conditions results in performance degradation and possibly instability. Therefore the mismatch should be minimized. If it is assumed that a larger prediction error results in worse performance of the system then the objective function in equation (15) maximizes the overall performance of the system.

Remark 3. In Proposition 1, the weighting parameter α_i is a design parameter by which the prediction error in each subsystem can be controlled. A higher weight is equivalent to a lower desired prediction error for that subsystem.

5.2.2.3 Discussion

The optimization problem discussed in Proposition 1 is based on the Bellman-Grown Lemma. However, due to the conservative nature of this lemma in finding an upper bound on the state prediction error, the increase in robust performance may not be fully optimal. Use of more accurate less conservative approaches for bounding the state prediction error, such as differential inclusions [20], can potentially improve the method. The proposed approach, however, has a clear advantage over more sophisticated computationally expensive bounds. The cost function in (15) for Proposition 1 should be modified accordingly, depending on the specific method used for finding the state prediction error bound.

5.2.3 Example

In this section, the proposed method described in Section 5.2.2.2 is applied to RHC stabilization of two double-integrator systems using a single computer. The dynamic equations describing a differentially driven wheeled mobile robot, or a rotorcraft-like UAV flying at constant altitude, can be transformed into two double integrators using feedback linearization [16], [17], [18]. Therefore, controlling a set of double integrators is potentially useful for autonomous formation guidance of multiple unmanned systems, where it is desired that each agent takes a predefined position [19].

The nominal system is described by the set of equations

$$\begin{cases} \dot{x}_{i,1} = x_{i,2} \\ \dot{x}_{i,2} = u_i \end{cases} \qquad i = 1,2 \tag{18}$$

where $x_{i,1}$, $x_{i,2}$ represent the states and u_i represents the input to the ith system. In the simulations, it is assumed that the actual system is described by the following equations

$$\begin{cases}\dot{x}_{1,1} = x_{1,2} \\ \dot{x}_{1,2} = u_1 + 0.5\sin(tx_{1,1})\end{cases}$$
$$\begin{cases}\dot{x}_{2,1} = x_{2,2} \\ \dot{x}_{2,2} = u_2 + 0.25\sin(tx_{2,1})\end{cases} \tag{19}$$

with initial conditions chosen to be $\boldsymbol{x}_0$=[9 -4]^T for the first subsystem and $\boldsymbol{x}_0$=[4 1]^T for the second subsystem. Among the available RHC methods, the quasi-infinite RHC approach described in [11] is implemented. The prediction horizon is selected as 5 seconds and the weighting matrices **Q** and **R** are taken as identity matrices for both subsystems. The matrix **P** in (4) is found by solving the Lyapunov equation (9) in reference [11].

If the 2-norm is used, the parameters found in the optimization problem discussed in equation (15) will take the following values: b_1=2.23, b_2=0.5 and L_1= L_2=1. The weighting parameters, α_i, and the upper bound on the execution horizon are selected as 1 and 0.25 respectively, for both subsystems. It is assumed that a zero control signal is applied to the system before the onset of RHC initialization. The optimization problem described in Proposition 1 was solved using MATLAB and the execution horizons were found to be δ_1=0.7965 and δ_2=0.8996. Therefore, using a rate monotonic approach, the highest priority is assigned to the first subsystem. The controllers for the first and second subsystems start the sampling at times of t=0.9465 and t=0.8996 respectively. This implies that before this time the controllers were in the off mode and no input was applied to the system. To investigate the pre-emption of the second subsystem by the first one, a 0.15 second delay for the start time of the first subsystem controller was applied (see Figure 5). Numerical simulations indicated that $\delta_{c,i} \leq 0.35$.

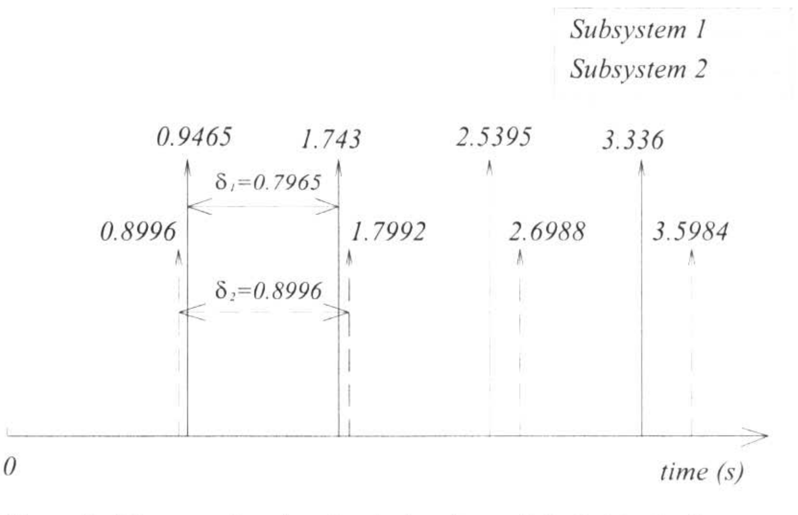


Figure 5- Diagram showing the timing for each individual subsystem

To solve the optimal control problem, the states and inputs were approximated by a set of piecewise polynomials of third order with second order continuity at the boundaries, which is an adaptation of the scheme described in [5]. In order to solve the resulting optimization problem, the NAG C Library (Mark 7) was used on a 2.4 GHz Pentium 4 computer. The Venturcom RTX (version 5.1) real-time operating system was used for implementation. Numerical simulation was carried out using the C programming language.

Two main functions used from NAG are "e04ucc" [36] and "e04xzc" [37][4].

The response of the closed-loop RHC system for the first and second subsystems is shown in Figure 6 and Figure 7 respectively. It is apparent that the states of both subsystems are stabilized around the origin despite the perturbations introduced to the system. It is also evident that the proposed scheduling approach achieves similar levels of performance for each subsystem even though subsystem 1 has significantly higher uncertainty than subsystem 2. This is a consequence of the higher priority allocated to subsystem 1 which improves its performance by reducing the computational delay compared to subsystem 2. The scheduling of the CPU time is shown in Figure 8. As shown in Figure 8, subsystem 2 starts the optimization process at t=0.8996 and is pre-empted by subsystem 1 at t=0.9465 due to the fact that subsystem 1 has a higher priority. Subsystem 2 is remained pre-empted until t=1.2818 and at this time the computation of subsystem 2 is started (from finishing computation of first subsystem until beginning computation of the second subsystem, the pre-emption overhead should be considered in timing). The computation of subsystem 2 is finished on t=1.5298. Therefore the computing time for subsystem 1 (with pre-emption overhead) is 0.3353 seconds and the total computing time for subsystem 2 (with pre-emption overhead) is 0.2948 seconds. The pre-emption process is repeated again between 6 and 8 seconds.

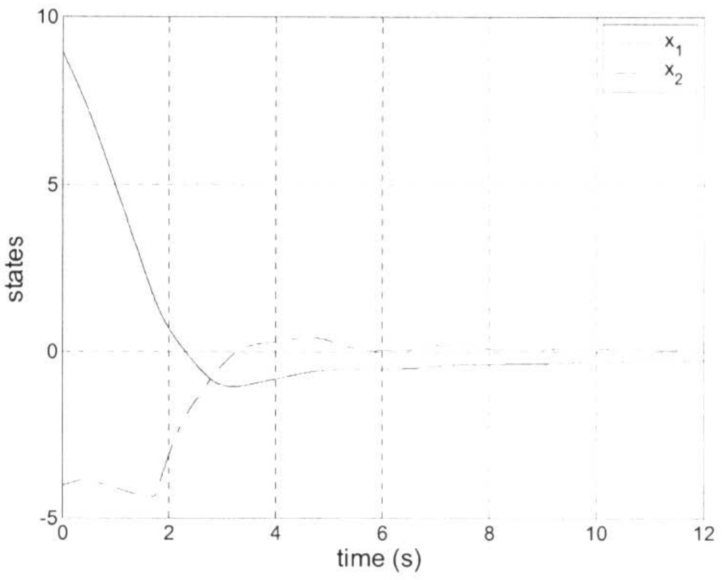


Figure 6- Time history of the states for the first subsystem

[4] Since this work (with NAG) is going to be redone using RHCOOL and SNOPT [38], no more explanation is done in this report.

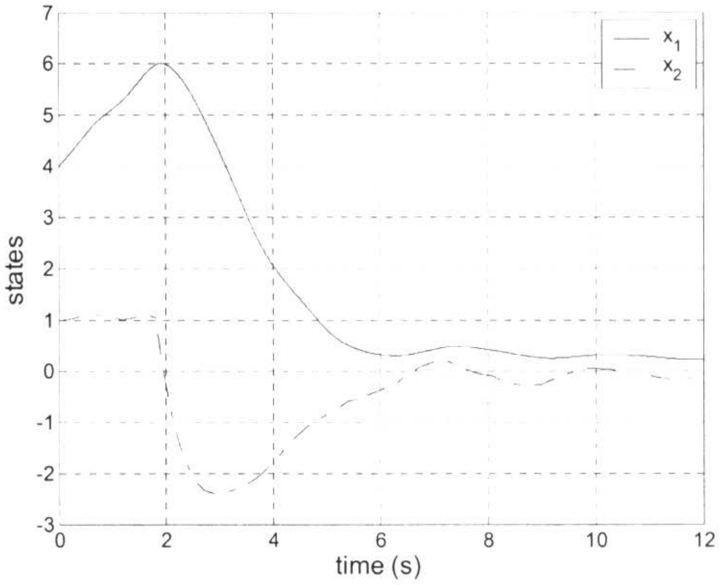


Figure 7- Time history of the states for the second subsystem

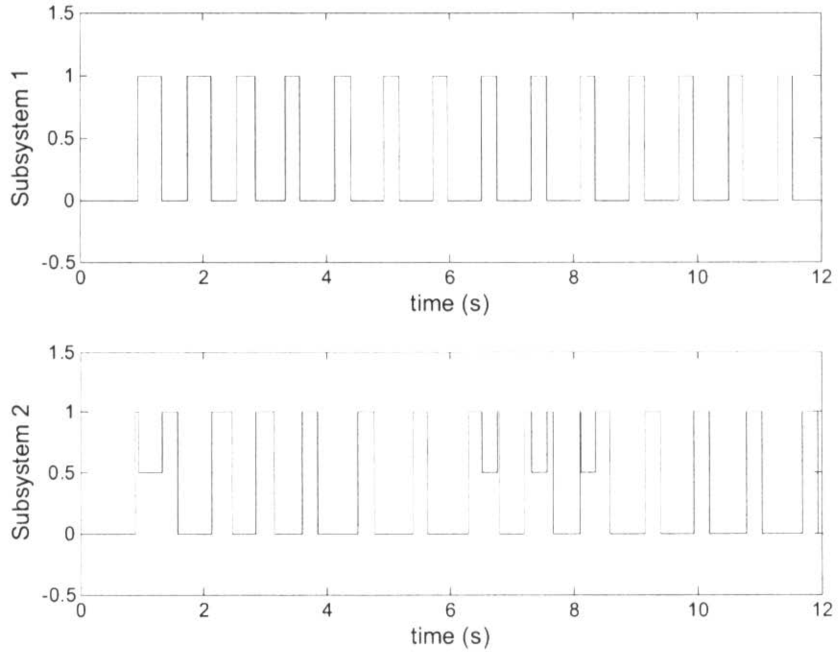


Figure 8- CPU time schedule (1=running, 0.5=Pre-empted, 0=Idle) for the first subsystem (top) and the second subsystem (bottom)

5.2.4 Discussion

In this section, a systematic approach is developed for scheduling computation and determination of the execution horizon for multiple uncertain RHC systems. It was shown that using a rate monotonic priority assignment method combined with analytical bounds on the prediction error, the problem of scheduling multiple uncertain plants can be cast into an appropriate constrained optimization problem. The constraints guarantee that the processes will be schedulable, and the optimization provides performance robustness to uncertainty. The proposed method was applied to a double integrator example problem demonstrating the validity of the approach.

In the next sections the proposed algorithm will be applied to a set of under actuated 3DOF miniature hovercraft. Experimental data ensures the performance of the proposed approach.

5.3 Application to 3DOF Miniature Hovercraft Problem

5.3.1 Modeling

The aim of this section and the next one is to identify the parameters of Miniature Hovercraft. Since RHC approach is Model Based, Finding these parameters are critical for implementing the RHC approach to the miniature hovercraft.
For obtaining the equations of motion of the miniature under actuated hovercraft, three coordinate frames are defined (Figure 9).

{G} is the Global coordinate frame (in Figure 9, the black axes X and Y denote for the Global coordinate frame.)
{B} is a rotating coordinate frame which the center of this frame is the center of {G} and the X_B axis is always in the same direction with the symmetric axis of hovercraft. (in Figure 9, the blue axes X_B and Y_B are illustrating the coordinate frame {B})
{M} is a moving coordinate which is attached to the hovercraft. This is defined in order to make the definition of {B} easier. The axis of {B} and {M} are always in the same direction but the center of these two coordinate frames are different. (in Figure 9, the red axes X'_B and Y'_B are illustrating {M}).

The variables which are used in this part are as follows[5]:

M	The total mass of hovercraft
J	The moment of inertia of hovercraft around Z axis
b_1	The coefficient of viscous friction in the forward movement
b_2	The coefficient of viscous friction in the side movement
b_3	The coefficient of viscous friction in rotation
F_1	Force caused by left side motor
F_2	Force caused by right side motor
u	Velocity of hovercraft along X_B axis
v	Velocity of hovercraft along Y_B axis
r	Angular velocity of hovercraft around Z axis
$\vec{V}$	Total velocity vector (Velocity in {G})
$\vec{a}_{H,G}$	Total Acceleration Vector (Acceleration in {G})

[5] As the variables are only used in this section and do not affect the whole report, they are not mentioned in the Nomenclatures in the beginning of the report.

l	Distance between the centers of two propellers

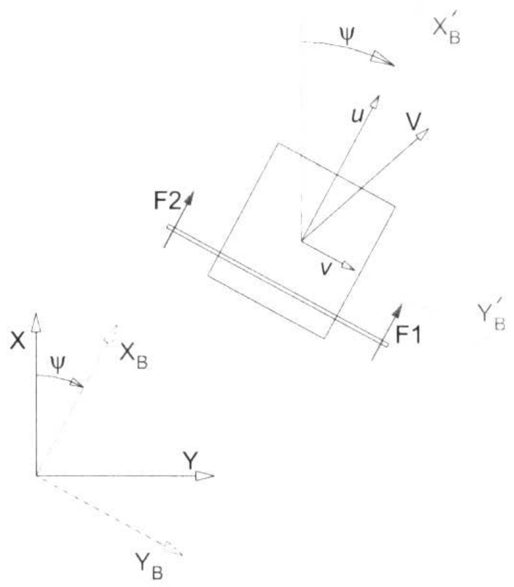


Figure 9- a schematic drawing of hovercraft and coordinate systems

The velocity of hovercraft in the Global coordinate frame {G} can be defined as follows:

$$\vec{V} = u\,\hat{i}_B + v\,\hat{j}_B \tag{20}$$

The acceleration can be defined by derivation of velocity vector as follows:

$$\begin{aligned} \vec{a}_{H,G} &= \dot{\vec{V}} \\ \dot{\vec{V}} &= \dot{u}\,\hat{i}_B + u\,\dot{\hat{i}}_B + \dot{v}\,\hat{j}_B + v\,\dot{\hat{j}}_B \end{aligned} \tag{21}$$

Since the {B} coordinate is rotating with the rate of r, the following equations can be used [26]:

$$\begin{aligned} \dot{\hat{i}}_B &= \vec{r} \times \hat{i}_B = r\,\hat{j}_B \\ \dot{\hat{j}}_B &= \vec{r} \times \hat{j}_B = -r\,\hat{i}_B \end{aligned} \tag{22}$$

Therefore the following equation can be derived from equations (20), (21), and (22)

$$\vec{a}_{H,B} = \dot{u}\,\hat{i}_B + ur\,\hat{j}_B + \dot{v}\,\hat{j}_B - vr\,\hat{i}_B = (\dot{u} - vr)\,\hat{i}_B + (\dot{v} + ur)\,\hat{j}_B \tag{23}$$

If no friction exists, from Newton's Second Law:

$M(\dot{u} - vr) = F_1 + F_2$ $M(\dot{v} + ur) = 0$ $J\dot{r} = {}^{l}\!/_{2}(F_2 - F_1)$	(24)

If viscous friction is considered we have the following equations:

$$M(\dot{u} - v\,r) = F_1 + F_2 - b_1\,u \tag{25}$$
$$M(\dot{v} + u\,r) = -b_2\,v \tag{26}$$
$$J\dot{r} = {}^{l}\!/_{2}(F_2 - F_1) - b_3\,r \tag{27}$$

where the b_1, b_2, and b_3 are the coefficients of viscous friction in X_B, Y_B, and rotation directions respectively.

Equations (25), (26), and (27) are the same as equations (2a), (2b), and (2c) in [22] respectively, except that in [22] the coefficients of viscous friction are considered to be the same for all directions.

As mentioned above, the aim of this section is to use the identified model in the RHC approach for performing the experimental verification. So the equations (25), (26), and (27) should be rearranged by performing some algebraic operations in order to:

- reduce the number of unknowns
- make the input signal (output of controller) compatible with the hardware

The first part is for simplicity in application of model identification and the second part is the main reason of model rearranging. In reality the controller output will be applied to the applied voltage of the motors of hovercraft. So it makes sense to use the applied voltages directly as the inputs. If the relation between the produced forces by propellers and the applied voltage to the motors can be considered linear[6], the reformed equations of motion will be as follows:

$$\dot{u} - vr = b_1' u + a_1 u_1 + a_2 u_2 \tag{28}$$
$$\dot{v} + ur = b_2' v \tag{29}$$
$$\dot{r} = b_3' r + a_3 u_1 + a_4 u_2 \tag{30}$$

[6] A better estimation for relation between the applied voltage and the produced forces by propeller, have been discovered, and best results were obtained when produced force was related to the linear composition of voltage and square of voltage.

In the equations (28) to (30), u_1 and u_2 are the normalized input voltages to the left and right motors and vary from -1 to 1. b_1', b_2', and b_3' are new forms of viscous friction coefficients and equal to $-{}^{b_1}\!/_{M}$, $-{}^{b_2}\!/_{M}$, and $-{}^{b_3}\!/_{M}$ respectively.

5.3.2 System Identification

In this section the unknown parameters of the model described in equations (28), (29), and (30) will be defined. The parameters that must be defined are $b_1', b_2', b_3', a_1, a_2, a_3$, and a_4.

In order to perform identification, the data from position and orientation of hovercraft are needed. The sensor that used for that purpose is PCIBird[7] [23] which has the capability of measuring Position and Orientation in 3D space with respect to a Global Co-ordinate frame. Therefore the data from PCIBird need two modifications:

- Coordinate transformation
- Obtaining velocities and acceleration

Coordinate transformation

Since the equations of motion are expressed in {B} the coordinate transformation is necessary. It should be noted that the equations dealing with velocities can be transformed using a rotation matrix. Equations (31) and (32) describe the results.

$$u = \dot{X}\cos\psi + \dot{Y}\sin\psi \tag{31}$$
$$v = -\dot{X}\sin\psi + \dot{Y}\cos\psi \tag{32}$$

But $\dot{u}$ and $\dot{v}$ can not be calculated using a rotation matrix. They must be derived by direct derivative of u and v, respectively because of their physical meanings. The results are as follows by derivation of equations (31) and (32):

$$\dot{u} = \ddot{X}\cos\psi - \dot{X}r\sin\psi + \ddot{Y}\sin\psi + \dot{Y}r\cos\psi \tag{33}$$
$$\dot{v} = -\ddot{X}\sin\psi - \dot{X}r\cos\psi + \ddot{Y}\cos\psi - \dot{Y}r\sin\psi \tag{34}$$

Obtaining velocities and acceleration

[7] The identification can also be done using the combination of Vision system and InertiaCube. These sensors will be explained in section 4. The main reason of using PCIBird is that the data measured by this sensor is more accurate than vision system and also all of the data can be measured by one sensor.

As briefly mentioned, the outputs of PCIBird [23] are only position and orientation, therefore velocity and acceleration should be calculated from the original position and orientation data. Since the hovercraft model is 2DOF only X, Y, and Yaw will be used. Velocity and acceleration should be calculated using the following "Finite Difference Method":

$$\dot{X}_n = \frac{X_{n+1} - X_n}{t_{n+1} - t_n} \tag{35}$$

where X_n and t_n denote for position (X axis) and time of the n^{th} sampling data. The same equation is used for other directions.

As a result of that, the noises in measurement can cause significant errors in calculated velocities and accelerations. To avoid the noise effect, the differentiation in equation (35) should be done in a larger time-step. So a revised form of equation (35) can be considered as follows:

$$\dot{X}_n = \frac{X_{n+filter} - X_n}{t_{n+filter} - t_n} \tag{36}$$

By selecting a proper *filter* value, the noise effect can be reduced and neglected. The following two steps are used in order to calculate u, v, r, $\dot{u}$, $\dot{v}$, and $\dot{r}$:

Step 1
From equation (36) the $\dot{X}$, $\dot{Y}$, and r values are calculated from the measured data (X, Y, and Yaw (ψ)). At this step *filter1* is used as the *filter* value. By using this data filtering, all of the data which are used in model identification must be updated. Therefore a new set of data will be produced in this step. Therefore parameters t and ψ will be updated using the following equation:

$$Var_{n,new} = \frac{1}{2}(Var_{n+filter} + Var_n) \tag{37}$$

where *Var* denotes for the parameter that have to be updated.

Step 2
From equation (36) the $\ddot{X}$, $\ddot{Y}$, and $\dot{r}$ are calculated from the calculated data in the first step. At this step *filter2* is used as the *filter* value. Same as step 1, another set of data must be calculated. Therefore $\dot{X}$, $\dot{Y}$, r, t, and ψ will be calculated from their values in the Step 1, using equation (37).

Step 3
From equations (31) to (34) u, v, $\dot{u}$, and $\dot{v}$ will be calculated.

5.3.2.1 Experimental results

In the following, the results of some different tests have been shown. In all of the following results the filter values were 100 and 70, respectively unless other specified.

5.3.2.1.1 Using input signal – Forward movement

In the following set of data, the same command is sent to the motors ($u_1 = u_2$) so the movement was mainly in the X_B (X axis of {B}). Consequently, it is better to use these results only for equation (28).
In this part the results of four data sets have been presented.

Data1:
In this case the input signal was 0.5 ($u_1 = u_2 = 0.5$) and the results were as follows:

$b_1^{'} = -0.29$

$a_1 = a_2 = 0.11$

The results are shown in Figure (10).

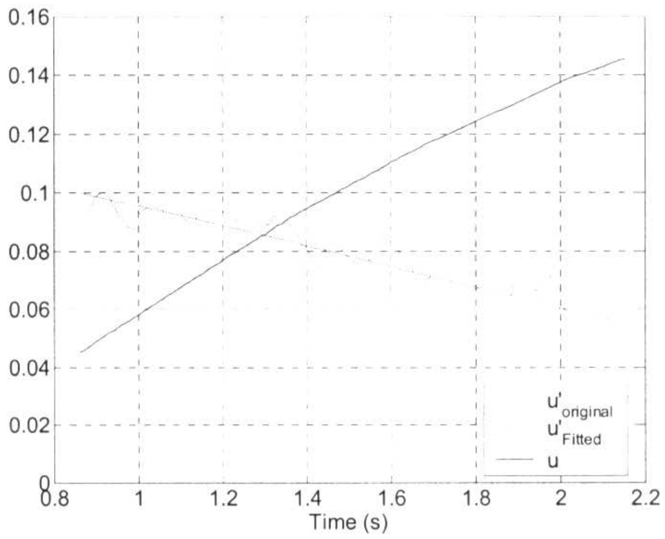


Figure 10- $u\,(m/s)$, $\dot{u}_{original}\,(m/s^2), and\ \dot{u}_{fitted}\,(m/s^2)$ are shown for *Data1* of *Part one*

Data4:
In this case the input signal was 0.7 ($u_1 = u_2 = 0.7$) and the results were as follows:

$b_1^{'} = -0.25$

$a_1 = a_2 = 0.13$

The results are shown in Figure (11). This result was not correct and the friction coefficient was more than zero which is not reasonable.

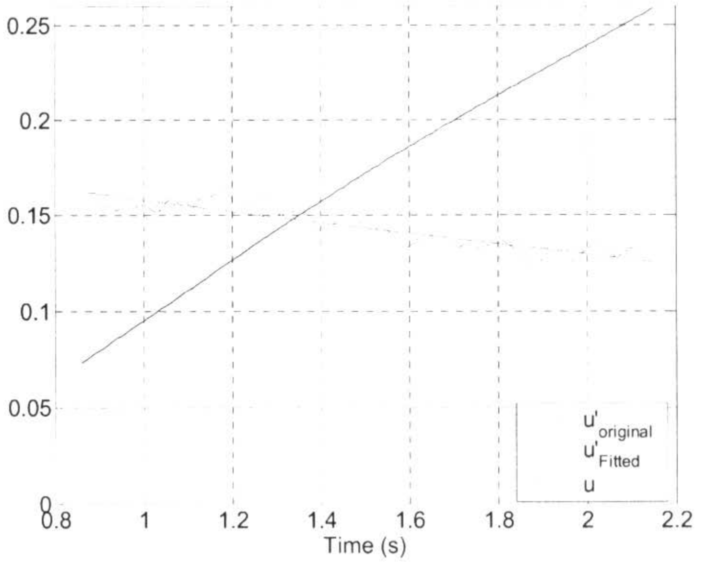


Figure 11- $u\,(m/s)$, $\dot{u}_{original}\,(m/s^2)$, *and* $\dot{u}_{fitted}\,(m/s^2)$ are shown for *Data4* of *Part one*

<u>*Data7:*</u>

In this case the input signal was 0.85 ($u_1 = u_2 = 0.85$) and the results were as follows:

$b_1' = -0.14$

$a_1 = a_2 = 0.12$

The results are shown in Figure (12).

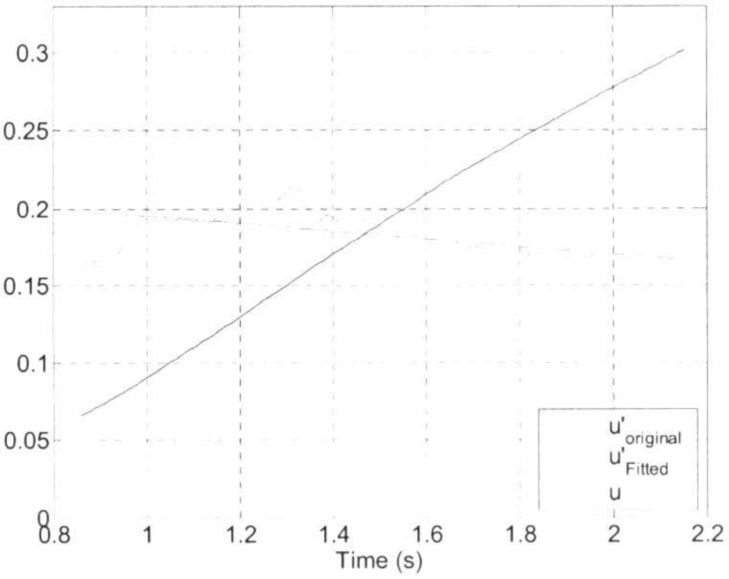


Figure 12- $u\,(m/s)$, $\dot{u}_{original}\,(m/s^2)$, *and* $\dot{u}_{fitted}\,(m/s^2)$ are shown for *Data7* of *Part one*

<u>*Data9:*</u>

In this case the input signal was 1.0 ($u_1 = u_2 = 1.0$) and the results were as follows:

$b_1^{'} = -0.07$

$a_1 = a_2 = 0.10$

The results are shown in Figure (13).

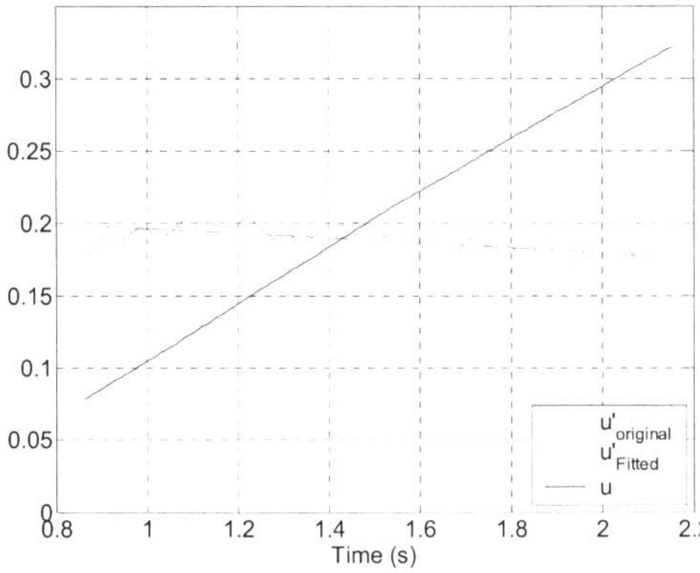


Figure 13- $u\,(m/s)$, $\dot{u}_{original}(m/s^2), and\, \dot{u}_{fitted}(m/s^2)$ are shown for *Data9* of *Part one*

5.3.2.1.2 Using input signal – Rotation

In this part the hovercraft was oscillating around zero angle. As a result, this data should be used only for equation (30). This oscillation was done by using a P controller (P=0.5) and only one set of data is shown.

Data2:

The results were as follows:

$b_3^{'} = -0.05$

$a_3 = -1.2$

$a_4 = 1.2$

The results are shown in Figure (14).

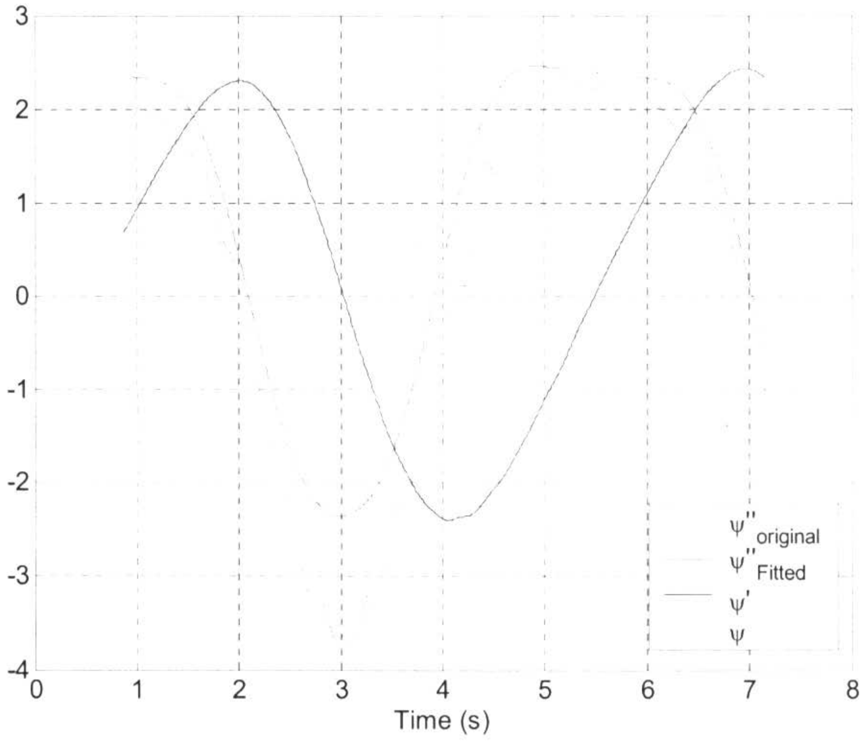


Figure 14- $\psi\,(rad),\ \dot{\psi}(rad/s),\ \ddot{\psi}_{original}(rad/s^2), and\ \ddot{\psi}_{fitted}(rad/s^2)$ are shown for *Data2* of *Part two*

5.3.2.1.3 No input signal

In this part, no signal applied to the motors and initial velocity was used, instead. The result of this test was used to identify the friction coefficients in the equations (28) and (29).

a) Calculating b_1' from equation (28)

It was varying between -0.13 and -0.56. Some of the graphs are shown here:

Data5:

The results were as follows:

$b_1' = -0.25$

The results are shown in Figure (15).

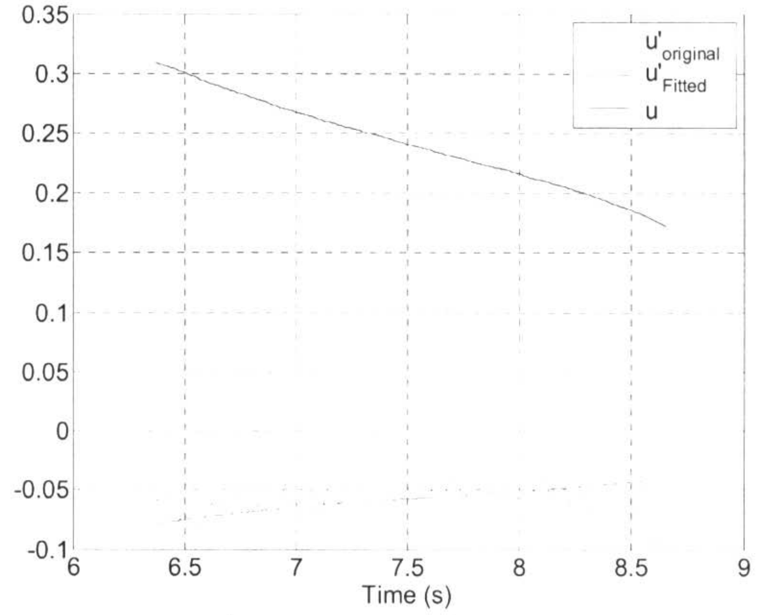


Figure 15- $u\,(m/s), \dot{u}_{original}\,(m/s^2), and\ \dot{u}_{fitted}\,(m/s^2)$ are shown for *Data5* of *Part three*

Data8:

The results were as follows:

$b_1' = -0.15$

The results are shown in Figure (16).

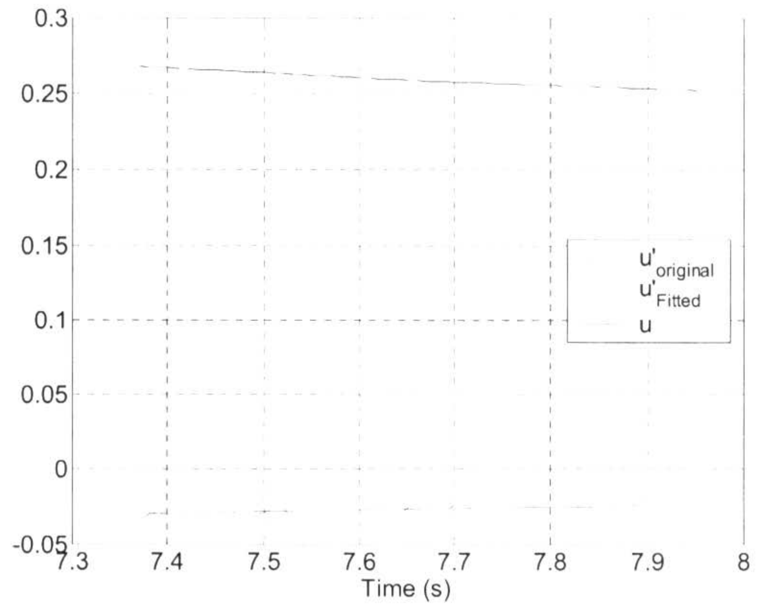


Figure 16- $u\,(m/s), \dot{u}_{original}\,(m/s^2), and\ \dot{u}_{fitted}\,(m/s^2)$ are shown for *Data8* of *Part three*

b) Calculating b_2' from equation (29)

It was varying between -0.12 and -0.02. One of the graphs is shown here:

Data2:

$b_2' = -0.02$

The results are shown in Figure (17).

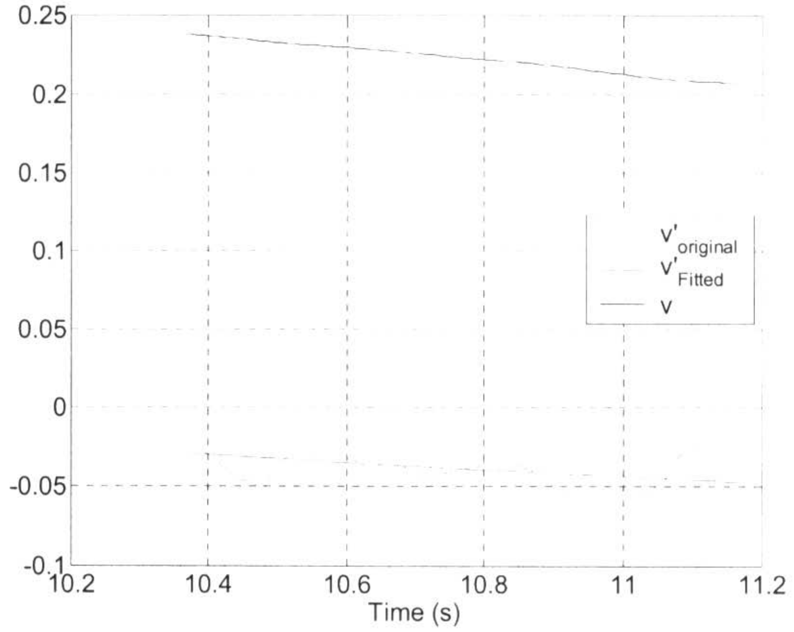


Figure 17- $v(m/s), \dot{v}_{original}(m/s^2), and\ \dot{v}_{fitted}(m/s^2)$ are shown for *Data2* of *Part three*

5.3.2.2 Discussion

From the presented identification algorithm, the following values have been defined. Some of the results are in the report.

b_1' :	-0.56, -0.30, -0.29, -0.25, -0.25, -0.24, -0.18, -0.15, -0.14, -0.13, -0.07, 0.03, 0.03, and 0.05. By reducing -0.56 and the positive values and averaging between the remaining data, $b_1' = -0.20$
b_2':	-0.12, -0.02. By averaging we have: $b_2' = -0.07$
b_3':	-0.18, -0.14, -0.05, 0.02, 0.08, 0.10, 0.13, 0.24, 0.24. By reducing the positive values and making average between the remaining data we have: $b_3' = -0.12$
a_1 :	0.07, 0.08, 0.10, 0.10, 0.11, 0.11, 0.12, 0.13, and 0.14. By averaging we have: $a_1 = 0.11$
a_2 :	same as a_1. So $a_2 = 0.11$

a_3:	-0.10, -1.02, -1.08, -1.13, -1.20. By making average we have: $a_3 = -1.1$
a_4:	same as -a_3. So $a_4 = 1.1$

5.3.3 RHC Object Oriented Library

In order to implement RHC to an experimental apparatus, you have to do different things like, Optimization and spline interpolation. In addition, to apply it to different apparatus and schedule them in one computer, it has to be object oriented. The RHC Object Oriented Library (RHCOOL) is developed in C++ and greatly simplifies the implementation of scheduled RHC hard real-time simulations and experimental implementations for multiple UAV systems. For implementing RHC to different apparatus by using the RHCOOL, only a few programming is required which are mainly for definition of model and constraints.

Following, first the library will be described and all of the parameters and functions which have been defined for implementation will be illustrated. After being familiar with the library, an example of 3DOF helicopter will be solved and described.

5.3.3.1 Description of the Library

The aim of this part is to describe the main functions and parameters of RHCOOL to a reader who wants to work with the library. We assumed that the reader is familiar with the basic specifications of RHC. In addition the user should be familiar with the basics of programming with C++.

There are six files in the library that the user should be familiar with. These files are *program.cpp (program_heli_local.cpp)*, *RHC_user.cpp (RHC_user_heli_local.cpp)*, *RHC.h*, *dVector.cpp*, *dMatrix.cpp*, and *dMatrix_array.cpp* which will be described in the following sections.

5.3.3.1.1 RHC.h

This file defines a class with name of *RHC* which all of the parameters and functions of the library are defined in it. RHC.h is available in the Appendix B. The *public* parameters of *RHC class* are as follows:

t0

t0 is a *double* variable and denotes the initial time. It is the time that controller is started and usually is equal to zero. It does not change during the optimization.

Remark: Figure (18) illustrates the different "times" used in the library.

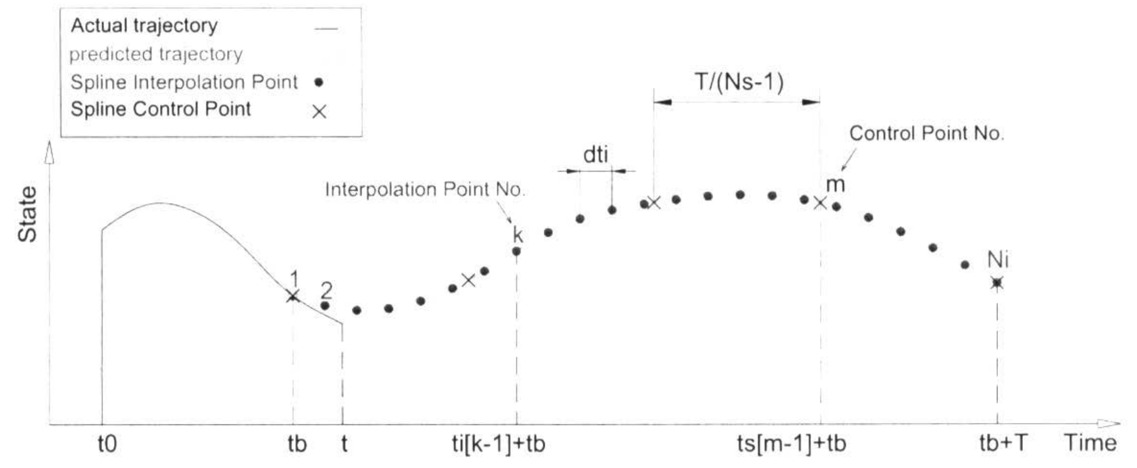


Figure 18- different "times" used in RHCOOL

t

t is a *double* variable and denotes the current time.

tb

tb is a *double* variable and denotes the time that interpolation begins. The RHC optimizes the outputs for the prediction horizon (T) and applies the result to the execution horizon (delta). Therefore each optimization will start at time *tb*, and end at time *tb+T*.

Remark:

It is useful to describe the library more. Figure 19 illustrates two systems, one is the input calculated by controller and the other is the input applied to Actual system. At time *tb*, the sampled data are ready and controller starts to calculate the input over the period of *tb* to *tb+T*. This calculation finished at time $tb+\delta_c$[8] and the calculated inputs are starting to be applied to the system at time $tb+T_p$. So the inputs applied to the system with a delay equal to the execution horizon (δ which is equal to the period (T_p) of the periodic task defined for the controller). One solution for this delay is using Retarded Actuation Technique, which is mentioned in section 2. In RHCOOL retarded actuation does not applied at this step for simplicity. So the calculated input in the period of *tb* to *tb+T* is should be applied to the system in the period of *tb+T* to *tb+2T* (the red lines in Figure 19 are the same). Another issue related to the RHCOOL is that the calculated inputs are applied to the system using First Order Hold approximation. So during the execution horizon, the input values do not change. It is not a big issue and is supposed only for simplicity and will be fixed in the next version of the library.

[8] δ_c is the computational time.

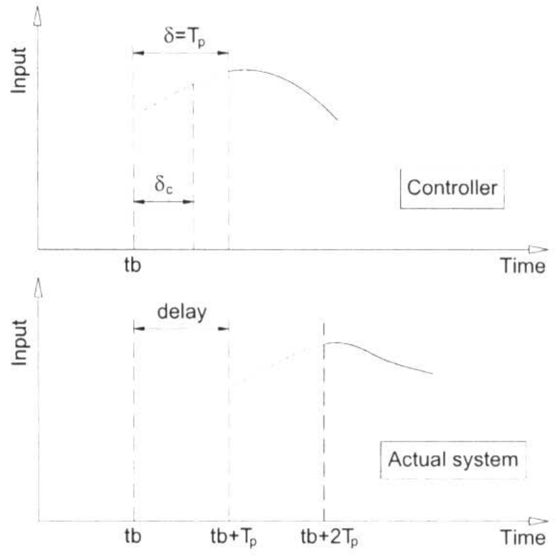


Figure 19- Illustration of *tb*, execution horizon ($\delta = T_p$), and computation time (δ_c).

***ti**

**ti* is a *dVector* pointer and denotes the interpolation time. When the interpolation is going on, the time at break points (interpolation points) of the spline[9] form the *ti* vector and each element of this vector refers to the time of each interpolation point. The elements of *ti* changing from 0 to (*Ni*-1)*dti*=*T.*

***ts**

**ts* is a *dVector* pointer and denotes the time at spline points. In addition to the statement about *ti*, the time in the control points of spline form the *ts* vector. The components of *ts* varying from 0 to T.

dti

dti is a *double* variable and denotes the interpolation time interval. It is equal to T/(Ni-1).

T

T is a *double* variable and denotes the RHC prediction horizon. During this time, the optimization is done.

delta (δ)

delta is a *double* variable and denotes the RHC execution horizon. The execution horizon is the time that calculated inputs from optimization are applied to the system.

Nx

Nx is an *int* variable and denotes the number of state variables.

[9] See [29] for information about cubic splines. This document is available in the CD attached to the report (c3-3.pdf).

Nu
Nu is an *int* variable and denotes the number of inputs.

Nz
Nz is an *int* variable and denotes the number of flat outputs. See [5] for more information about flat outputs.

Ny
Ny is an *int* variable and denotes the number of outputs.

*X0
**X0* is a *dVector* pointer and denotes the initial condition of state variables.

*U0
**U0* is a *dVector* pointer and denotes the initial inputs.

*X
**X* is a *dVector* pointer and denotes the current state (at time t). The number of components of this vector is equal to *Nx*.

*U
**U* is a *dVector* pointer and denotes the current input. The number of components of this vector is equal to *Nu*.

*Y
**Y* is a *dVector* pointer and denotes the current output. The number of components of this vector is equal to *Ny*.

*Z
**Z* is a *dVector* pointer and denotes the current flat output. The number of components of this vector is equal to *Nz*.

*Xd
**Xd* is a *dVector* pointer and denotes the current state derivatives.

Ns
Ns is an *int* variable and denotes the number of spline control points for each output. Number of control points is similar to the degree of polynomial. As discussed later, the varying control points are the parameters which will be find by optimization.

Ni
Ni is an *int* variable and denotes the number of spline interpolation points for each output. After finding the parameters of spline by optimization, the values of that spline will be recorded in the interpolation point. Therefore the interpolation points (sometimes called spline break points) change the continuous spline curve to discrete values (make a

tabulated table). Ni should be large enough to make interpolation points close enough to each other.

Nc
Nc is an *int* variable and denotes the number of varying control points. Az mentioned above, all of the control points are in two main groups; one group is the fixed control points because of the constraint and the other group forms the varying control points. The second group will be defined by optimization.

Nd
Nd is an *int* variable and denotes the maximum number of output derivatives required. For calculating state variables and inputs from flat outputs, the derivatives of the flat outputs are required. Nd, is the maximum derivative required from all of the flat outputs.

Ks
Ks is an *int* variable and denotes the order of bspline (must be at least Nd + 1). This is used when bspline used for interpolation.

***Cs**
Cs is a *dMatrix* pointer and denotes the spline control points. All of the spline control points form the matrix *Cs*. This matrix is in the form of *Cs[q][j]*. *q* is the related flat output and *j* is the related control point.

***Cv**
Cv is a *dVector* pointer and denotes the vector containing all varying control points. All of the varying control points from all of the flat outputs form the vector *Cv*. The definition of this vector (how the control points of different flat outs are placed) is defined by user in function *calc_J* of the *RHC_user.cpp* (*RHC_user_heli_local.cpp*).

***Xi**
Xi is a *dMatrix* pointer and denotes the state variables *Xi[j][i]* for each interpolation point *i*. As mentioned before, all of the values of state variables, inputs, outputs, and flat outputs are recorded in the interpolation points. So the matrix *Xi*, is for all of the values of state variables which are the result of the last optimization over the prediction horizon.

***Ui**
Ui is a *dMatrix* pointer and denotes the input *Ui[j][i]* for each interpolation point *i*. The definition of *Ui* is similar to *Xi*, and represents all of the values of inputs which are the result of the last optimization over the prediction horizon.

***Yi**
Yi is a *dMatrix* pointer and denotes the output *Yi[j][i]* for each interpolation point *i*. The definition of *Yi* is similar to *Xi*, and represents all of the values of output variables which are the result of the last optimization over the prediction horizon.

***Zi**

Zi is a *dMatrix_array* pointer and denotes all of the flat outputs and their derivatives at each interpolation time (*Zi[q][id][i]*). *q* (1 to *Nz*), is for each flat outputs, *id* (0 to *Nd*) is for each derivative and *i* is for each interpolation time.

***Zi0**
Zi0 is a *dMatrix* pointer and denotes flat output value at the beginning of the interpolation time *tb* (*Zi0[q][id]*). *q* (1 to *Nz*), is for each flat outputs and *id* (0 to *Nd*) is for each derivative.

***ZiT**
dMatrix ZiT[q][id]
ZiT is a *dMatrix* pointer and denotes flat output values at the end of the interpolation time *tb+T* (*ZiT[q][id]*). *q* (1 to *Nz*), is for each flat outputs and *id* (0 to *Nd*) is for each derivative.

***Q, *R, and *P**
These are cost function parameter matrices and are pointers in the form of *dMatrix*. Q is a *Nx* by *Nx* matrix, R is a *Ny* by *Ny* matrix, and P is a *Nx* by *Nx* matrix.

***Bs**
Bs is *dMatrix_array* pointer and denotes the bspline basis functions. *Bs[id][i][j]* for derivative id (0 to *Nd*), interpolation time i (1 to *Ni*), and control point j (1 to *Ns*). This is used when the bspline is used in interpolation.

***C2**
C2 is a *dMatrix* pointer and denotes the second derivative at control points. It is arranged in the form of *C2[q][j]* for flat output *q* and control point *j*. It is used in cubic spline interpolation method.

Imethod
Imethod is an *int* variable and denotes the interpolation method (1 = cubic spline, 2 = bspline, 3 = 4th order continuity cubic spline).

tout_final
tout_final is a *double* variable and denotes the final output time.

Nout
Nout is an *int* variable and denotes the number of output points.

Nsamp
Nsamp is an *int* variable and denotes the number of output samples

***Xout**
Xout is a *dMatrix* pointer and denotes the state *Xout[j][i]* for each time *tout[i]*. The result of simulating state variables is stored in this matrix.

*Uout

Uout is a *dMatrix* pointer and denotes the input *Uout[j][i]* for each time tout[i]. The input which is applied to the system (real or virtual model) is stored in this matrix.

*Yout

Yout is a *dMatrix* pointer and denotes the output *Yout[j][i]* for each time *tout[i]*. The result of simulating outputs is stored in this matrix.

*Zout

Zout is a *dMatrix* pointer and denotes the flat output *Zout[j][i]* for each time *tout[i]*. The result of simulating flat outputs is stored in this matrix.

*tout

tout is a *dVector* pointer and denotes the output time with spacing at the interpolation points *dti*.

contructor function

RHC(double t0, double T, double delta, int Nx, int Nu, int Nz, int Ny, dVector &x0, dVector &u0, int Ns, int Ni, int Nc, int Nd, double tout_final, int Imethod, int ks)

func_powell

This used in optimization with Powel''s method.

calc_J

This function calculates the cost function for trajectory generation. It is described in 3.3.1.3 and 3.3.2.2.

optimize_cost

This function, optimizes the cost function and is described in 3.3.1.2 and 3.3.2.2.

interpolate

This function does the interpolation and it is done automatically be the library.

calculate_XU

This function calculates *Xi* and *Ui* based on the flat outputs *Zi*. It is described in section 3.3.1.3.

calculate_U

This function calculates the input at time *tc*. It is also described in section 3.3.1.3.

simulate_interval(double tf, double dt)

This function simulates the system from the current time *t* to the final time *tf* using the optimal input *Ui* and a time step *dt*. The results are stored in the output variables *Xout*, *Uout*, *Yout*, and *tout*.

calculate_xd

This function calculates the derivative of the state variables (xd = f(x,u,t)) of the actual system. It is used in simulation.

calculate_outputs
This function calculates the outputs from the state variables.

save_data
This function saves variables into a text file. It is described in section 3.3.1.2.

save_flat_outputs
This function saves flat output variables over the current interpolation interval into a text file.

5.3.3.1.2 program_heli_local.cpp[10]

This file includes the "main" function and the following steps are done in it:

a) Variable definition
The variables which are used in RHC class definition are defined in this first stage. The basic variables can be classified in the following manner:

- Key dimensions: *Nx*, *Nu*, *Nz*, *Ny*, and *Nd*
- Design variables: *Ns*, *Ni*, *T*, *delta*, *Imethod*, and *ks*
- Number of free optimization parameters: *Nc*

b) Declaring the RHC objects
Using the variables defined in the first part, the RHC objects are defined. We can define one or more than one objects here.

c) Setting Initial Conditions
There are two ways to apply initial conditions.

- One way is to apply them before declaring the RHC objects. In this case the initial conditions should be applied to *X0*.
- The other way is to apply them to *X*, the current state vector, after declaring the RHC objects.

d) Optimizing the cost function over one horizon
The main purpose of this step is for debugging and understanding how good one optimization horizon is. This can be done by saving the data[11] and plotting the results in the Prediction Horizon (*T*). Furthermore, this may be good for tuning the parameters as well as good for checking how the flat outputs describe the system.

[10] The code is available in appendix C.

[11] This is also done in the program and the result is saved in "f.dat". The command line in the "*program_heli_local.cpp*" is: `rhc1.save_flat_outputs("f.dat");`

e) Simulating 3dof helicopter in local coordinates using RHC approach
For simulation two main functions will be executed:

- `rhc1.optimize_cost();`[12]
- `rhc1.simulate_interval(t,rhc1.dti/1000);`

The first one is the main function which performs the optimization over the specified prediction horizon and the second one is used to do the simulation.

Before applying the controller to the real apparatus it is wise to test it on a virtual system and perform some basic tuning. This is done by using the second function. Therefore the second function is used instead of applying the controller to a real apparatus.

It must be noted that the optimization is done over each execution horizon (*delta*) therefore the simulation is also done for that time. Furthermore the user can specify the actual model that is used for the simulation in RHC_user (calculate_xd). This can be useful to test the robustness to uncertainty. The *dt* for the simulation is made smaller than *dti* (*T*/(*Ni*-1)) to get a more accurate simulation using Euler's method (here *dt* = *dti*/1000).

f) Saving the simulation output
The function "`save_data("file_name")`" saves the output data at time intervals of dti[13] to a file. The data is saved in the following pattern:

- time (first column)
- states (second column to column # 1+Nx)
- inputs (column # 2+Nx to # 1+Nx+Nu)
- flat outputs (column # 2+Nx+Nu to # 1+Nx+Nu+Nz)
- outputs (column # 2+Nx+Nu+Nz to # 1+Nx+Nu+Nz+Ny)

5.3.3.1.3 RHC_user_heli_local.cpp[14]

All of the functions that need changing in future because of the application to different apparatus are in this file1. These functions will be explained in the following:

calculate_U
The input of this function is *Time* and calculates the input at that *Time*, based on the inputs which are calculated from the optimization.
From optimization, the inputs on the spline interpolation points (spline break points) are known, so the inputs are known only in some points and it is not a continuous function.

[12] This function is explained in section 3.3.2.2.
[13] The data are saved at each interval times of *dti*, so there is not too much data.
[14] The code is available in appendix D.

calculate_U perform a linear interpolation[15] between the known values of inputs and the inputs will be continuous functions of time.

calculate_xd
This function is useful for simulation. Using *f_heli_3dof2* function, it calculates the derivatives of state variables. Note that by using a model in helicopter in *f_heli_3dof2* function, which is different from the model used in controller, the robustness of controller can be investigated.

calculate_outputs
This function calculates the outputs from the state variables.

calculate_XU
This function is one of the main functions in this file. It calculates Xi (state variables) and Ui (inputs) based on the Zi (flat outputs). Since, some of the state variables and inputs are calculated from equations of motions, the model (equations of motion) and the model parameters are included in this function and must be changed if the model or parameters change.

calc_J
This function is the most important function in this file. It calculates the cost function for trajectory generation. The model parameters, all of the constraint including the initial conditions, and the specifications of the desired trajectory must be defined here. The input of this function is a *dVector* containing all of the varying control points. The following steps are done in this function:

- Getting control points from optimization vector c (input) and forming the Cs vector for the first flat output.
- Applying initial conditions to the Cs vector, so the control points vector of the first flat output will be completely defined.
- Defining boundary conditions of the spline
    - Zi0: boundary condition derivatives (including zero derivative) at t=0
    - ZiT-- boundary condition derivatives (including zero derivative) at t=T
- Performing the foresaid three steps to the other flat outputs
- Spline interpolation which is done by function "interpolate". The RHCOOL will do interpolation itself and do not need any modifications.
- Calculating Xi and Ui based on Zi. This is done by function "calculate_XU" which described before.
- Integrating cost function by trapezoidal integration method. The cost function is defined here. Therefore if the result of controller is not satisfying one way is changing the cost function.

f_heli_3dof2

[15] As the interpolation points are usually close to each other, the result of linear interpolation is acceptable. However, it is also possible to make spline interpolation but the results are usually the same while linear interpolation take less computation time.

This function has the 3DOF model of helicopter from [6] and computes the derivatives of the state variables. This function is described completely in section 3.3.2.

5.3.3.1.4 dVector.cpp[16]

This file defines a class with name "*dVector*". *dVector* defines a one dimensional array (vector) based on dynamic memory allocation. The basic format for defining a variable in the form of *dVector* is:

```
dVector var(n1,n2)
```

which defines a variable with the name of *var*. *var* is a one dimensional array of *double* variables. *n1* and *n2* are indicating the first and last elements of the array, respectively. Furthermore var is initialized and all of its elements are zero.

For referring to the element #n the following command must be used:

```
var.e(n)
```

The benefits of using *dVector* are:

- It uses dynamic memory allocation without the common problems of using pointers.
- A basic one dimensional array in C++ starts from zero index, but with dVector it can be started from any integer, including positive and negative, numbers.
- The basic mathematical operations like adding, subtracting and multiplying are included in the function.

dMatrix.cpp[17]

This file is similar to the *dVecor* but instead of one dimensional array it defines a 2D array (matrix). This file defines a class with name "*dMatrix*". *dMatrix* defines a 2D array based on dynamic memory allocation. The basic format for defining a variable in the form of *dMatrix* is:

```
dMatrix var(n1,n2,m1,m2)
```

which defines a variable with the name of *var*. *var* is a two dimensional array of *double* variables. *n1* and *n2* are indicating the first and last rows of the array, respectively and *m1* and *m2* are indicating the first and last columns of the array. Furthermore *var* is initialized and all of its elements are zero.

[16] The code is available in appendix E.
[17] The code is available in appendix F.

For referring to the element in row #n and column #m the following command must be used:

```
var.e(n,m)
```

The benefits of using *dMatrix* are the same as *dVector*.

dMatrix_array.cpp[18]

This file is similar to the *dMatrix* but instead of 2D array it defines a 3D array. This file defines a class with name "*dMatrix_array*". *dMatrix_array* defines a 3D array based on dynamic memory allocation. The basic format for defining a variable in the form of *dMatrix_array* is:

```
dMatrix_array var(k1,k2,n1,n2,m1,m2)
```

The other specifications are the same as *dMatrix*.

5.3.3.2 RHC approach of 3DOF miniature helicopter using RHCOOL

In this section the 3DOF miniature helicopter is controlled using RHCOOL. First the equations of motion of the foresaid helicopter are described, and then the programs are explained.

5.3.3.2.1 Equations of motion of 3DOF miniature helicopter

The equations of motion of a 6DOF miniature helicopter are derived in [6]. Figure (20) illustrates the forces and moments acting on the helicopter and the coordinate system used to present the equations of motion. For 3DOF, the equations are reduced from the original 6DOF and presented in the same body coordinate of figure (20):

[18] The code is available in appendix G.

$$
\begin{aligned}
\dot{u} &= vr - \frac{1}{m} C_x u \sqrt{u^2+v^2} - \frac{k_{mr_x}}{m} T_{mr} \\
\dot{v} &= -ur - \frac{1}{m} C_y v \sqrt{u^2+v^2} \\
\dot{r} &= \frac{-l_{tr}}{I_{zz}} T_{tr} \\
\dot{\psi} &= r
\end{aligned}
\tag{38}
$$

where:

$$
\begin{aligned}
C_y &= \frac{1}{2} \rho_a S_{fus_y} \\
C_x &= \frac{1}{2} \rho_a S_{fus_x}
\end{aligned}
\tag{39}
$$

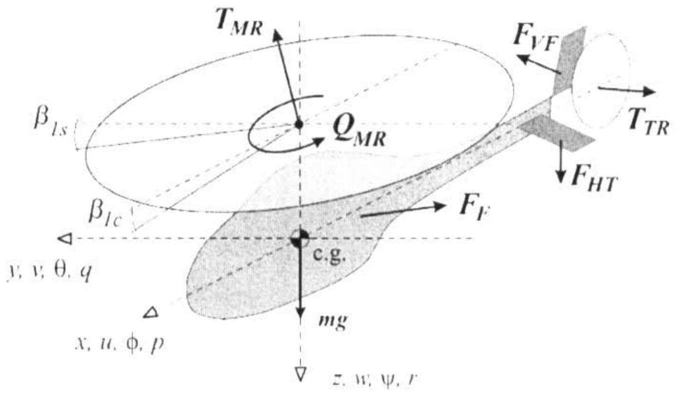


Figure 20- Moments and forces acting on the 6DOF helicopter and the used body coordinate frame [6]

Since, in this example, path following is considered, a global coordinate system is also used. For simplicity, the coordinate systems used for 3DOF hovercraft in section (3.1) (Figure 9) are the same as coordinate systems defined for 3DOF helicopter. The extra equations used for global coordinate is as follows:

$$
\begin{aligned}
\dot{X}_c &= u\cos(\psi) - v\sin(\psi) \\
\dot{Y}_c &= u\sin(\psi) + v\cos(\psi)
\end{aligned}
\tag{40}
$$

Based on the foresaid equations, the following assumptions are considered:

- u, v, r, ψ, X_c, and Y_c are considered as State Variables.
- C_y in equation (38) is supposed to be zero. This means that the friction in side direction considered to be zero for simplicity

- Since the friction in side direction is equal to zero, u and ψ are considered to be Flat Outputs [5].

Remark: the above assumption ($C_y = 0$) has the benefit that only two flat outputs (u and ψ) are enough to solve the problem. Because from equation (38), $\dot{v} = -ur$, so the state variable v can be defined by integration and does not need to add another flat output to the system. If $C_y \neq 0$, the number of flat outputs will be 3 (u, v, and ψ) which increase the computation time about 50 percent.

5.3.3.2.2 Example

In this section the know-how of using RHCOOL is described by implementing RHC to the 3DOF helicopter. The reader is encouraged to read the section 3.3.1 especially section 3.3.1.2. Also he/she should be familiar with the variable definition in section 3.3.1.1. The following steps are done in the main loop:

Step 1: Calculating the inputs by optimizing cost function (`optimize_cost`)
Step 2: Simulating the results (`simulate_interval`)
Step 3: Saving the results (`save_data`)

Each function (step) is described bellow:

Step 1: optimize_cost
The RHC will be executed just by calling "*optimize_cost*" in the main function. This function must be in a loop or in a periodic function. By executing this function two main tasks will be done:

- Calling "*calc_J*" which is a function for calculating cost function and forming the optimization variables
- Performing optimization by using Powell's method

Function *calc_J* is a key point in the library and the following four basic steps are done for calculating the cost function J[19]:

Step 1: *defining Cs (control points matrix) from endpoint constraints and varying control points*

The user does this and then RHCOOL modifies the spline control points appropriately.

[19] The user is encouraged to read *calc_J* function in section 3.3.1.3

For all of the control points there is a matrix (Cs). Some of the elements of this matrix must be defined by constrains of the problem and the other elements are from the varying control point vector which is the input of *calc_J*.

One of the most important parts that the user has to do is making a vector of free control pointsI). The parameters of this vector are the output of optimization. In this example, we have only two flat outputs, u and ψ, so the varying control point vector will be only from these two variables. This vector is configured such that the first elements are from u and the second elements are from ψ ($C = [u_{free}, \psi_{free}]$).

The possible constraints that considered in this example are only for the end points. So for each flat output, the variable and its derivatives must be checked at the start point and end point of the spline. If any constraint exists, its control point is fixed, and if there is no constraint, its control point will be a varying control point. For clarity, in this example the first control point of u will be fixed (u at time t0 has constraint) but the derivative of u does not have any constrain and its control point will be a varying one.

Step 2: *spline interpolation*
The library (RHCOOL) does this automatically. The user just call function "*interpolate*" after defining *Cs*.

Step 3: *calculate the State variables (Xi) and Inputs (Ui) based on Flat outputs (Zi)*
This part should be written by user for each RHC problem. This is done by calling function "*calculate_XU* ".
As mentioned in section (3.3.1.3), *calculate_XU* calculates Xi (state variables) and Ui (inputs) based on the Zi (flat outputs). Since, some of the state variables and inputs are calculated from equations of motions, the model (equations of motion) and the model parameters are included in this function and must be changed if the model or parameters change. This function is strait forward and completely depends on the equations of motion.

Step 4: *integration of cost function*
The user writes this function to Integrate cost function by trapezoidal integration method[20]. The cost function is also defined here. Therefore if the result of controller is not satisfying one way is changing the cost function.

Remark: the properties of the reference trajectory also must be defined in the function *calc_J*. In this example an ellipse is considered as the reference trajectory.

Step 2: simulate_interval

In this example, function "*simulate_interval*" is used to simulate the results the time equal to execution horizon[21]. Every parts of this function will be done by the library

[20] The user can integrate the cost function by other methods that he/she prefer but in this example trapezoidal integration is used.
[21] See section 3.3.1.2 for more information

except the modeling. The model and parameters which are used for simulation are in function "*f_heli_3dof2*". The equations of motion used in this function are equations (38) and (39). In addition, the parameters are the same as parameters defined in [6].

Step3: Save_data
The results are saved with the following manner: Time, states, inputs and flat outputs. Please see item *f* of section 3.3.1.2 for detailed information.

5.3.4 Real-time Scheduling of Hovercraft Problem

In this section, the proposed method described in Section 5.2.2.2 is applied to RHC stabilization of two hovercrafts using a single computer. Two models are considered in this section; one model deals with only rotation direction and the other has the 3DOF hovercraft.

Rotation direction only

In this case inputs to the motors of hovercraft are the same but in the opposite direction. Also from identification procedure the coefficients a_3 and a_4 of equations (30) was almost the same. Therefore the equations of motion for hovercraft # *i* can be estimated as follows:

$$\begin{aligned} \dot{r}_i &= b'_{i,3}r + 2a_{i,3}u_{i,1} \\ \dot{\psi}_i &= r_i \end{aligned} \tag{41}$$

Among the available RHC methods, the quasi-infinite RHC approach described in [11] is implemented. The state variables are ψ_i and r_i and the flat outputs [5] are ψ_i. The prediction horizon is selected as 1 seconds and the weighting matrices **Q** and **R** are taken as identity matrices for both subsystems. The matrix **P** in (4) is found by solving the Lyapunov equation (9) in reference [11]. The experimental results will be presented in section 4.

Full 3DOF Hovercraft

The equations of motion are from equations (28) to (30) and equations (40) for both apparatus. Therefore for subsystem # *i* the following equations are obtained:

$$\begin{aligned} \dot{u}_i &= v_i r_i + b'_{i,1}u_i + a_{i,1}u_{i,1} + a_{i,2}u_{i,2} \\ \dot{v}_i &= -u_i r_i + b'_{i,2}v_i \\ \dot{r}_i &= b'_{i,3}r_i + a_{i,3}u_{i,1} + a_{i,4}u_{i,2} \end{aligned} \tag{42}$$

$$\dot{\psi}_i = r_i$$
$$\dot{X}_{c,i} = u_i \cos(\psi_i) - v_i \sin(\psi_i)$$
$$\dot{Y}_{c,i} = u_i \sin(\psi_i) + v_i \cos(\psi_i)$$

The state variables are u_i, v_i, r_i, ψ_i, $X_{c,i}$, and $Y_{c,i}$. Among the available RHC methods, the quasi-infinite RHC approach described in [11] is implemented. The prediction horizon is selected as 1 seconds and the weighting matrices **Q** and **R** are taken as identity matrices for both subsystems. The matrix **P** in (4) is found by solving the Lyapunov equation (9) in reference [11].

As discussed in section 3.3.2 the $b'_{i,2}$ is considered to be zero. This assumption has this benefit that the number of flat outputs is reduced to two for each subsystem (u_i and ψ_i). By this assumption, $\dot{v}_i = -u_i r_i$, so the state variable v_i is calculated by numerical integration[22]. If $b'_{i,2}$ is not considered to be zero, another flat output (v_i) has to be added and the number of flat outputs will be increased to 3 which increase the computation time about 50 percent.

The experimental results will be presented in section 4.

[22] In section 3.3.2 trapezoidal integration was used for 3DOF helicopter.

5.4 Experimental Verification

In this section the result of applying RHC to single and multiple hovercrafts is investigated. It should be mentioned that in the section 3.3.2, a 3DOF helicopter model is solved by using the library in order to explain library and to notify the reader that this library is applicable to different apparatus. One of the future applications of this library is to apply RHC to multiple 6DOF helicopters and the example of 3DOF helicopter in addition of explaining the library, notify the applicability of the library to helicopters.

Furthermore, the equations of motion for a 3DOF helicopter are similar to equations of motion of hovercraft. Let's explain this similarity more. Equations (25) to (27) are for hovercraft and equations (38) are for 3DOF helicopter. In these equations there are two differences:

1- The friction terms are different which can be modified easily in the code and also in low velocities the differences are not so much.
2- The inputs to the system are different but with a transformation matrix they easily can be changed to each others as follows:

$$\begin{bmatrix} T_{mr} \\ T_{tr} \end{bmatrix} = \begin{bmatrix} \dfrac{-m}{M\,k_{mr_x}} & \dfrac{-m}{M\,k_{mr_x}} \\ \dfrac{l\,I_{ZZ}}{2J\,l_{tr}} & \dfrac{-l\,I_{ZZ}}{2J\,l_{tr}} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{43}$$

But the state variables are the same. In addition, all of the tests done in this report are with low velocities, so for the considered model in the controller, the friction coefficients are considered to be zero.

In the following sections, first, the apparatus is described and every component is explained, after that the open loop response is presented. Then the results of RHC for a single hovercraft and multiple hovercrafts are presented.

5.4.1 Description of the Apparatus

Figure 21 shows a schematic diagram of the setup. The setup consists of the following parts[23]:

1- A set of cameras (C1 to C4) used to capture images and to forward to the Vision computers (V1 to V4). Each image consists of some colored squares and the

[23] The experimental setup which is explained here, is the total setup prepared in the CIS Lab. However, some parts of the setup were used in each experiment.

computers calculate the centroid of each square (Figure 22). Each camera is connected to one computer in order to decrease the image processing time.

2- All of the centeroids calculated by the vision computers are transferred to the server via a Gigabyte Ethernet Switch. The server combines the centroid information and sends them to the main controller computers (M1 and M2) via another Gigabyte Ethernet Switch.

3- M1 and M2 receive the data from sensors. One sensor is the vision system and the data is sent from the server, and the other sensor is a wireless orientation sensor (Wireless InertiaCube3 [24]) and calculate the input. The input signals are transferred to the apparatus via data acquisition board and wireless transmitter[24] (T1 and T2). In this setup each computer is connected to two apparatuses, at most.

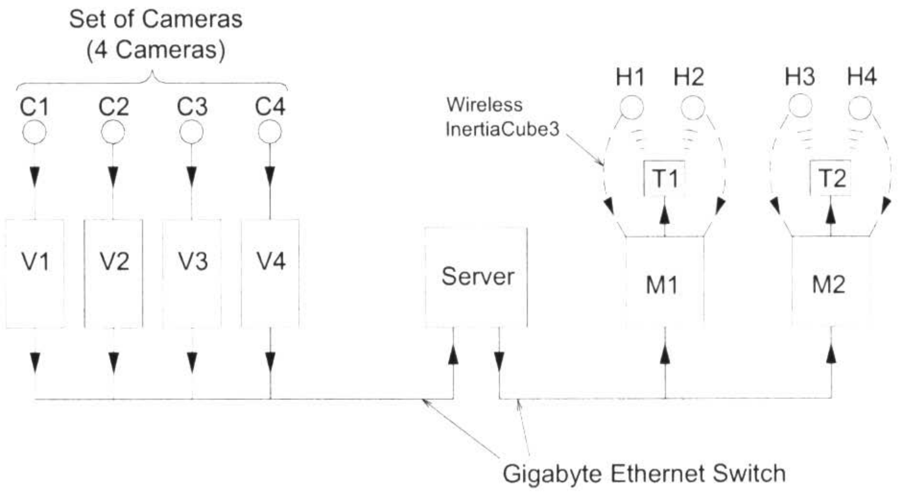


Figure 21- Schematic layout of experimental setup

[24] The wireless transmitter is "HiTec Laser 6 Channel 75MHz RC Controller Pkg" and is explained in [28] and [31].

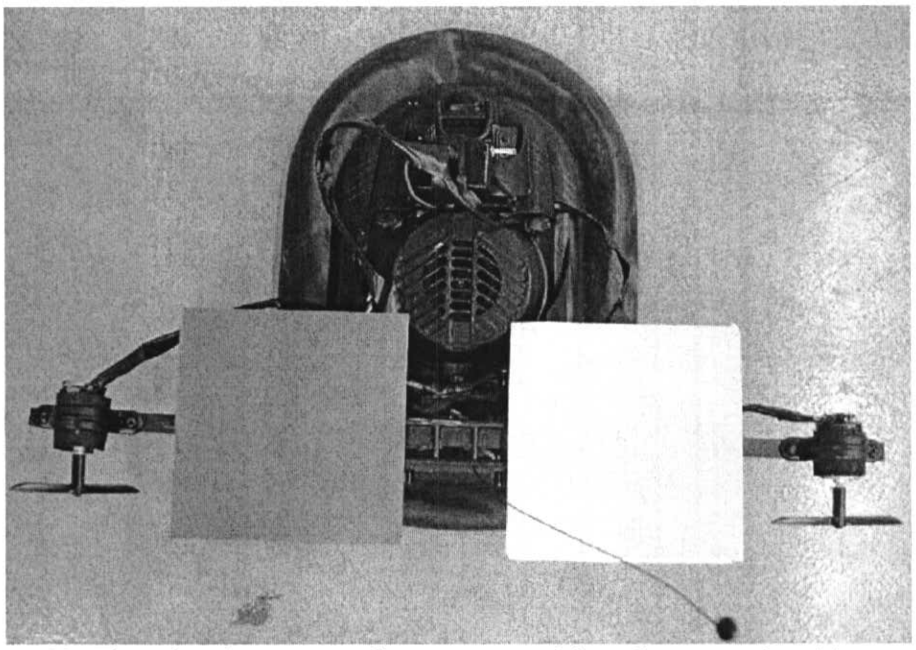


Figure 22– The colored squares used to measure position of apparatus by vision system

The apparatus used in this investigation is a miniature hovercraft and is shown in Figure 22. This hovercraft is a modified version of the R/C hovercraft [32]. The motors are placed with more distance from each other to produce a larger torque. In addition, a voltage amplifier board with a potentiometer is added. The voltage amplifier board is from an R/C motor, model HS-700BB [33].

For controlling the apparatus, a wireless transmitter is attached to the computer using a data acquisition board and the receiver is attached to the apparatus. In order to connect the transmitter to the data acquisition board some modifications are done in the transmitter. The data acquisition board, transmitter and all of the modifications are described in [28]. Two batteries are used as follows:

1- One 9.6 V battery used for the main fan which cause flouting of hovercraft. This fan pushes air under the hovercraft and causes it to move on a film of air, therefore the friction of hovercraft in all directions (forward, side, and angular) is very small.
2- One 6 V battery used for both propellers. This battery is connected to the receiver and both propeller motors are connected to the receiver.

As mentioned above, in order to measure orientation and angular velocity of miniature hovercraft, a wireless sensor is used. This sensor is a Wireless InertiaCube3 from InterSence [24]. The position in X-Y coordinate is obtained using Vision System which is developed in the CIS Lab. Theses two sensors are illustrated in the following two sections.

InertiaCube3 Sensor

In order to have orientation and angular velocity of miniature Hovercraft and miniature Helicopter, Wireless InertiaCube3 from InterSence has been used. This section describes the sensor.

5.4.1.1.1 Principles of InertiaCube3

The InertiaCube3 is an inertial 3-DOF (Degree of Freedom) orientation tracking system. It obtains its motion sensing using a miniature solid-state inertial measurement unit, which senses angular rate of rotation, gravity, and earth magnetic field along three perpendicular axes. The angular rates are integrated to obtain the orientation (yaw, pitch, and roll) of the sensor. Gravitometer and compass measurements are used to prevent the accumulation of gyroscopic drift [25].

The InertiaCube3 is a monolithic part based on micro-electro-mechanical systems (MEMS) technology involving no spinning wheels that might generate noise, inertial forces, and mechanical failures. The InertiaCube simultaneously measures 9 physical properties, namely *angular rates*, *linear accelerations*, and *magnetic field* components along all 3 axes. Micro-miniature vibrating elements are employed to measure all the angular rate components and linear accelerations, with integral electronics and solid-state magnetometers. *The magnetometers are included for optional yaw drift correction in the sourceless inertial orientation mode only.* (See Figure 23) [25]

Figure 24 illustrates the basic physical principal underlying all Coriolis vibratory gyros. Suppose that the tines of the tuning fork are driven by an electrostatic, electromagnetic, or piezoelectric drive to oscillate in the plane of the fork. When the whole fork is rotated about its axis, the tines will experience a Coriolis force $F = \vec{\omega} \times \vec{V}$ pushing them to vibrate perpendicular to the plane of the fork. The amplitude of this out-of-plane vibration is proportional to the input angular rate, and is sensed by capacitive, inductive, or piezoelectric means to measure the angular rate [25].

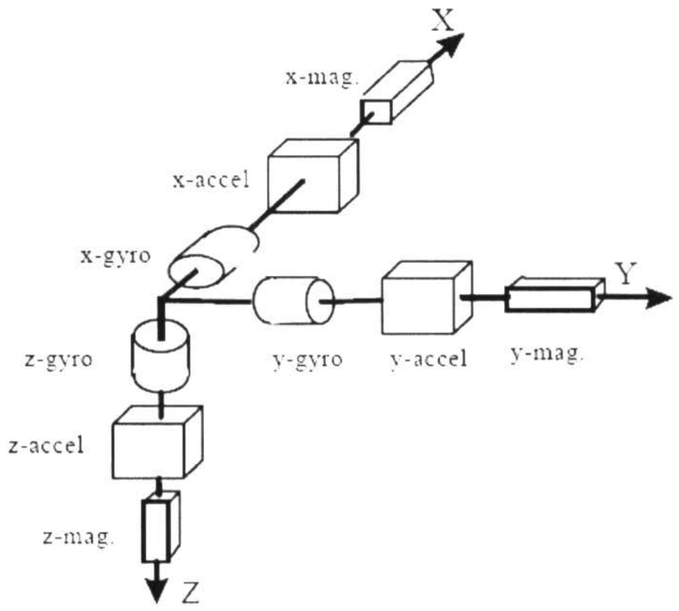


Figure 23 – Functional diagram of InertiaCube3 [25]

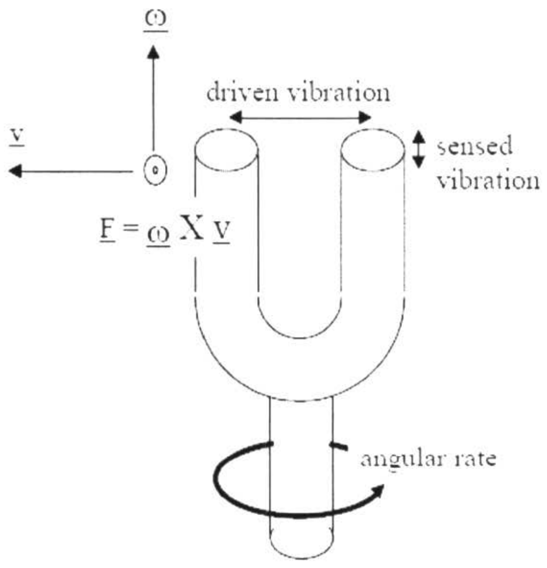


Figure 24- Principle of Coriolis vibratory Gyroscope [25]

5.4.1.1.2 Experimental data

As mentioned above, the InertiaCube3 can get orientation and angular velocities in 3D and present the information in the form of Euler Angles (Yaw, Pitch, and Roll). For a 3DOF hovercraft, only information in Yaw direction will be used. To prevent drift problem, this sensor is used with full compass mode and in this case any metallic objects, magnetic or electric field can cause inconsistency in sensor readings if they are in the vicinity of the sensor.

In order to know, how this sensor is working in CIS Lab (the area which is prepared for the vision system), some visual tests have been done and it was obvious that the sensor reading was not reliable and was changing in an arbitrary manner. So, the following tests have been done.

The sensor was moving along each line of B1 to B5 (Figure 25 and Figure 26) while was being kept in one direction. When this sensor was so close to the floor, the changing was more than having a higher level. So these tests were performed in three different levels.

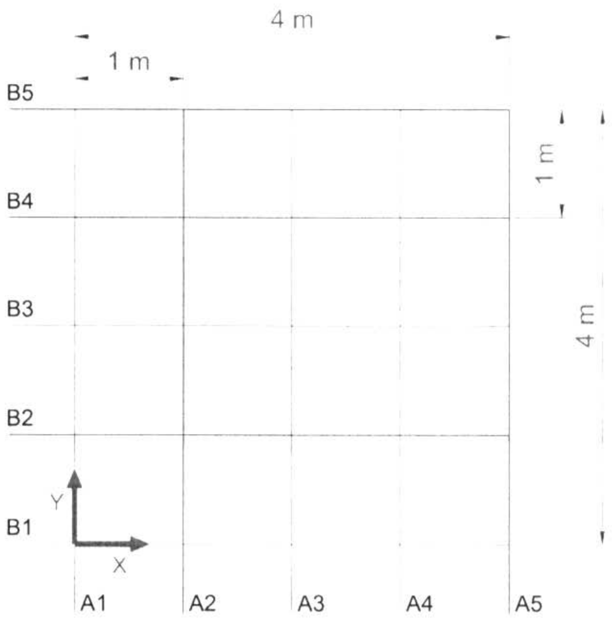


Figure 25- Used layout for getting data in CIS Lab

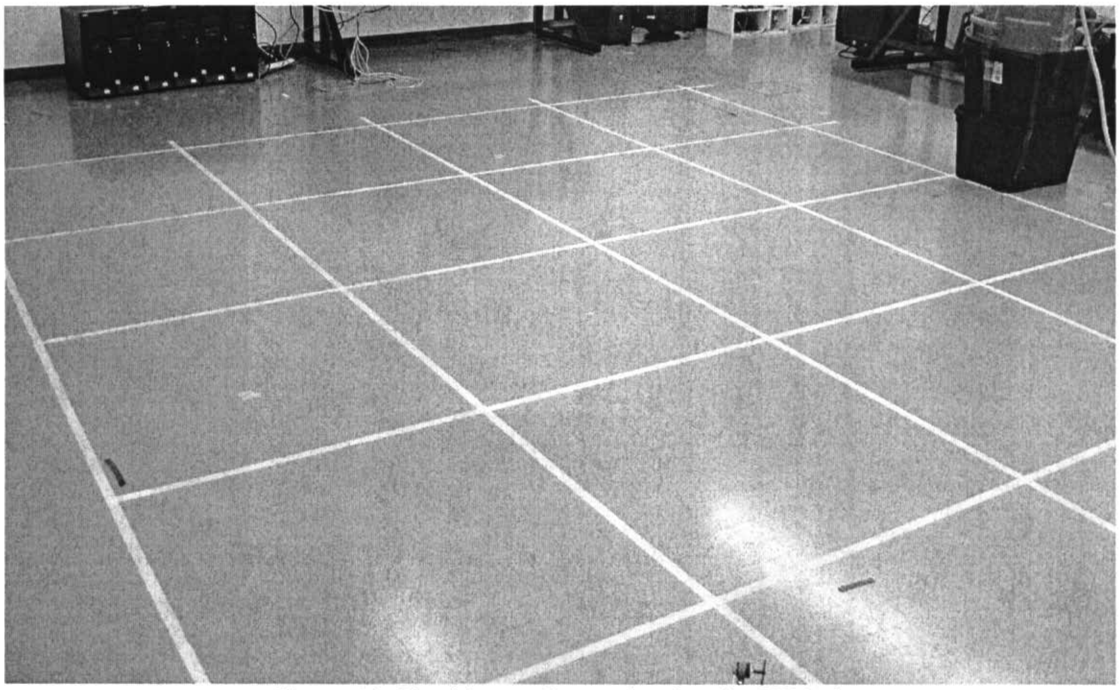


Figure 26- Used layout for getting data in CIS Lab

Level 1

In this level, the sensor was attached to a wood beam and has 3.5 cm distances from the floor (see Figure 27). The data are shown in Figure 28.

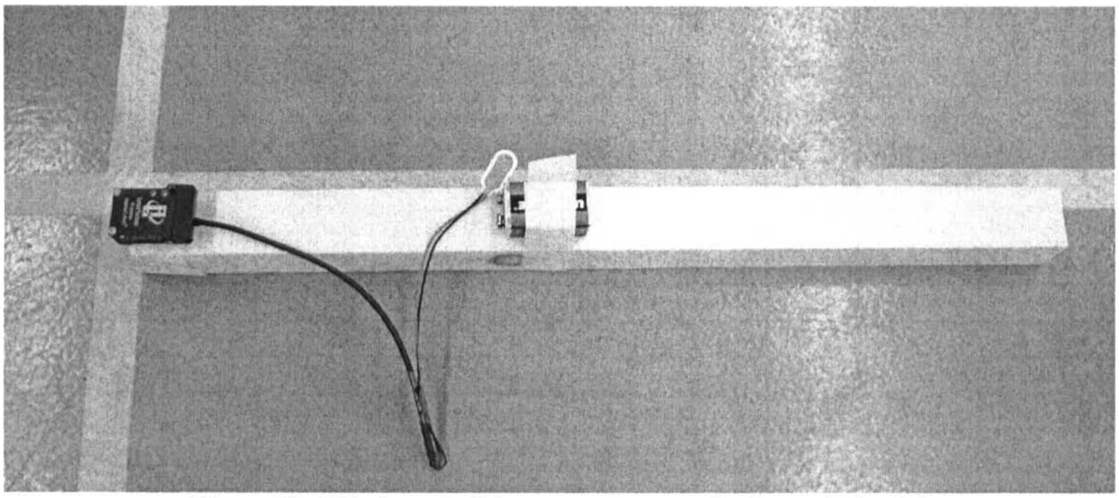


Figure 27- Attachment of the sensor to a wooden beam in level 1

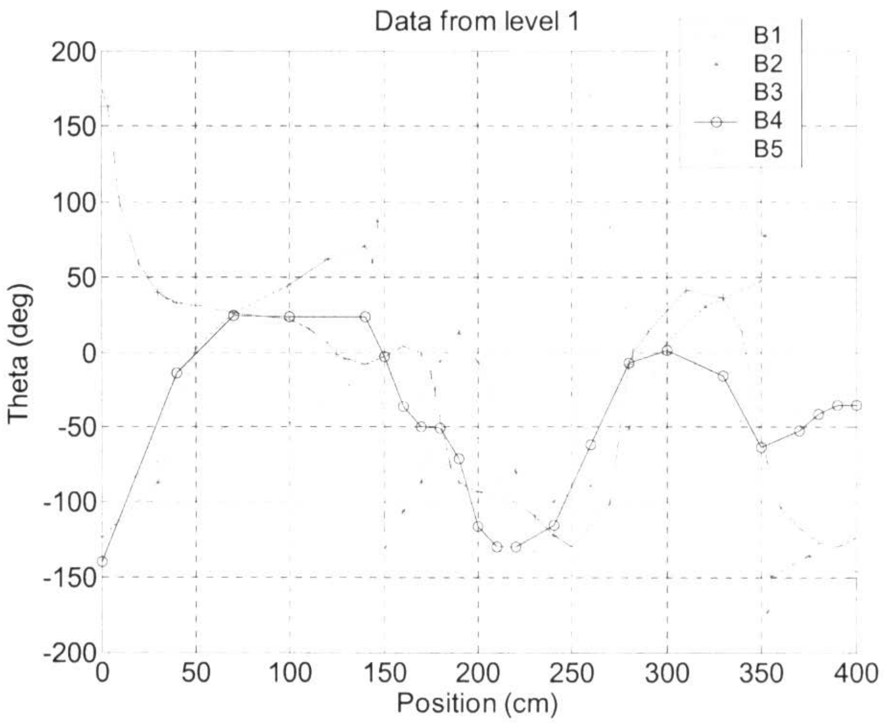


Figure 28- changing in sensor reading of Yaw direction when was kept 3.5 cm over the floor

In Figure 28, sensor was kept in *one level* (3.5 cm above the floor) and in the *same direction* and was following each lines of B1 to B5. If the area was perfect and without any magnetic field (except the Earth magnetic field), all of the data from sensor should be the same.

Level 2

In this level, the sensor was attached to a wood beam on top of a plastic box and has 65 cm distance from the floor (see Figure 29). The data are shown in Figure 30.

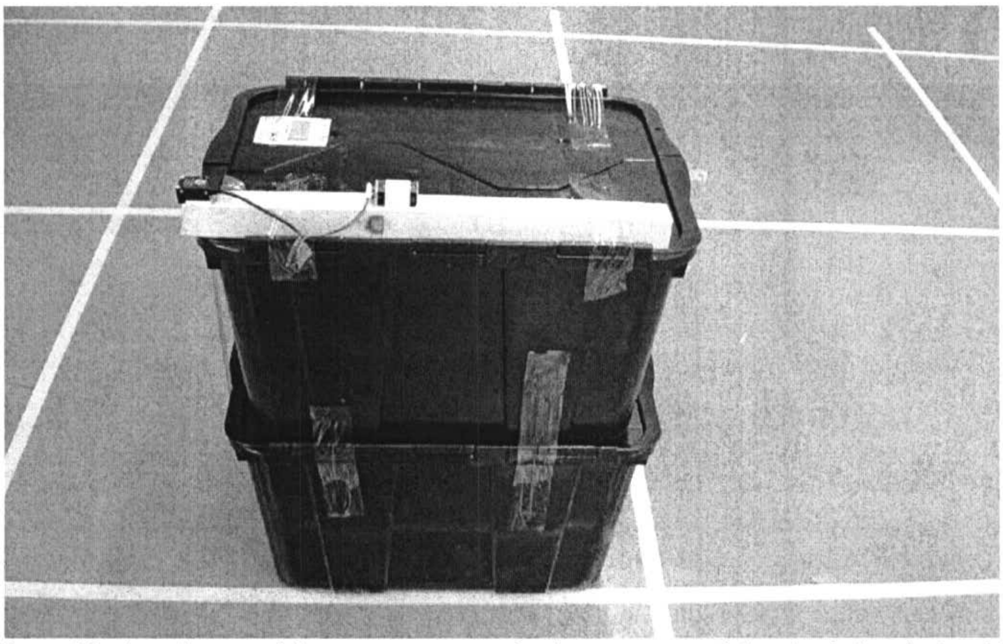


Figure 29- Attachment of the sensor to a wooden beam on top of the plastic box in level 2

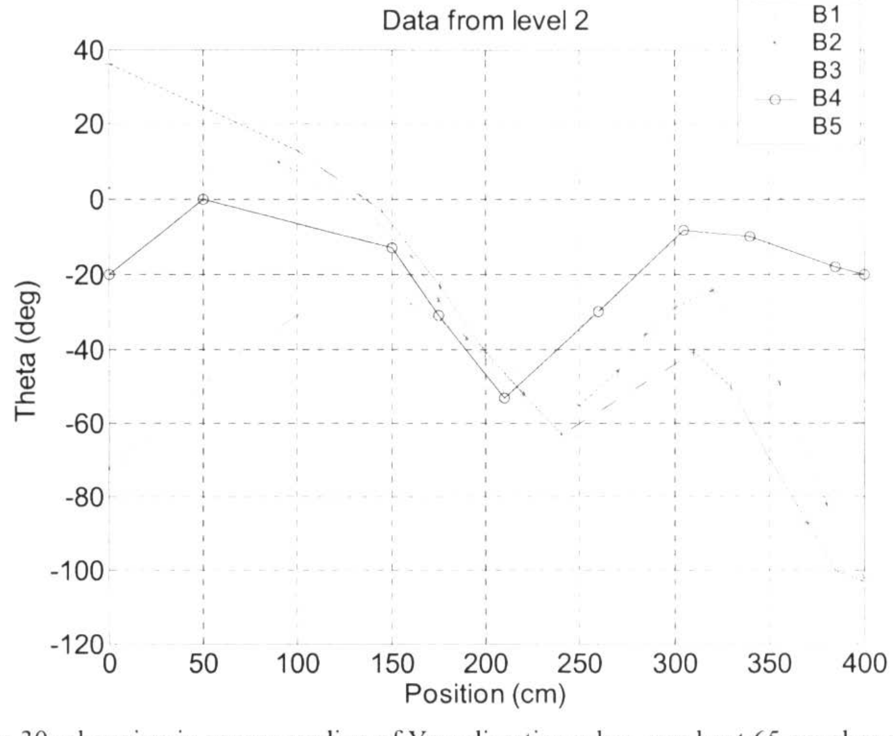


Figure 30- changing in sensor reading of Yaw direction when was kept 65 cm above the floor

In Figure 30, sensor was kept in *one level* (65 cm above the floor) and in the *same direction* and was following each lines of B1 to B5. If the area was perfect and without any magnetic field (except the Earth magnetic field), all of the data from sensor should be the same.

Level 3

In this level, the sensor was attached to a wood beam on top of a plastic box and has 125.5 cm distance from the floor (see Figure 31). The data are shown in Figure 32.

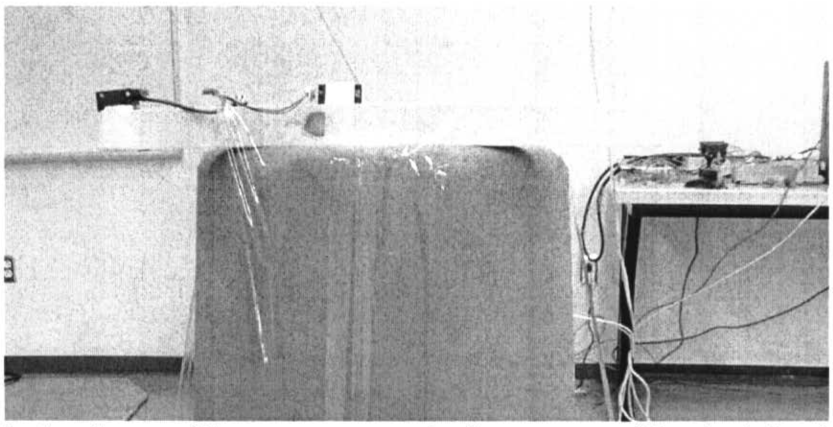


Figure 31- Attachment of the sensor to a wooden beam on top of the plastic box in level 3

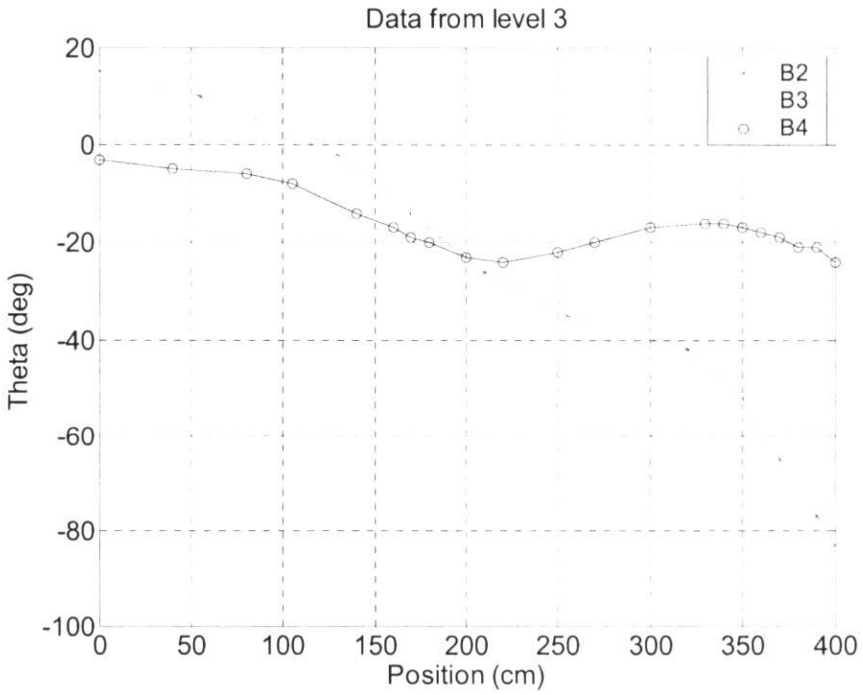


Figure 32- changing in sensor reading of Yaw direction when was kept 125.5 cm above the floor

In this case that changing is not too much, and it is possible to find an area with acceptable performance. So the data in A2 to A5 direction were recorded as well. (Figure 33)

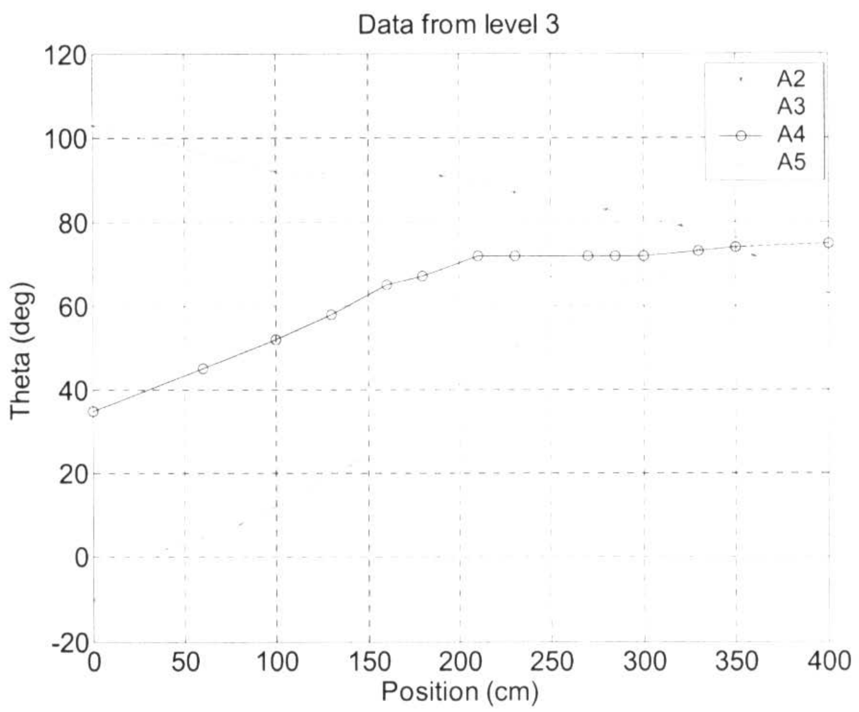


Figure 33- changing in sensor reading of Yaw direction when was kept 125.5 cm over the floor

5.4.1.1.3 Discussion

From Figure 32 and Figure 33 it can be shown that there exist some areas that this sensor can be used in. In the *area1* which is shown in Figure 34, the changes in sensor data are limited in 10 degrees. Also area2 in Figure 35 is a bigger area which changes are not more than 15 degrees.

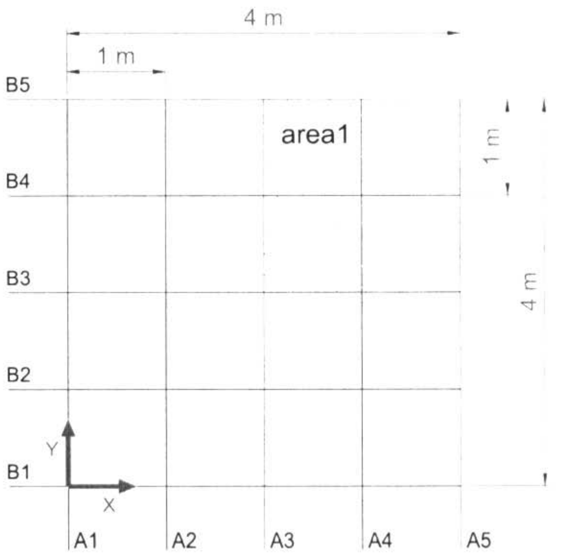


Figure 34- area which changes in sensor data is less than 10 degrees

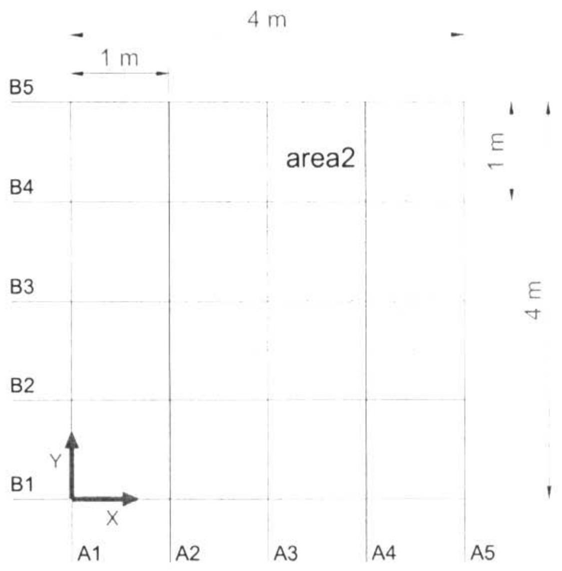


Figure 35- area which changes in sensor data is less than 15 degrees

So one way of working with this sensor in CIS Lab with Full Compass Mode, is having a table made of non-metallic materials with the height of about 125 cm.

The other solution can be shielding the area or using a larger magnet in order to make a unidirectional magnetic field in the area of working.

The other solution is to use the sensor "without Compass" or "Partial Compass Mode" (see [25] for more information). In this case the drift problem in the reading values of Yaw should be corrected with another solution. Based on this solution, the combination of wireless InertiaCube3 and vision system[25] is used in sections 4.3 and 4.4. So the angular velocity was measured by wireless InertiaCube3 and the angle was obtained from vision system.

Another thing which I should mention here is that the wireless InertiaCube3 is not sensitive to changing of the hovercraft batteries (The InertiaCube2 was sensitive to this matter and after changing the batteries it needed about half an hour being in the new situation to work properly).

5.4.1.2 Vision Array

In order to have the position (X-Y) of the hovercraft a vision array is built at a height of 3.5 meters in the test area of the CIS Lab. It produces a test area in the shape of a rectangle with dimensions of 4 by 5 meters. 4 webcams[26] are connected to 4 computers

[25] See section 4.1.2 for more information.

[26] Installing more webcams is also possible but with four webcams the working area is covered with a reasonable resolution.

and they send the data to a main computer which acts as a server. See Figure 21, and Figure 36 to Figure 38. Figure 36 shows an overview of the vision array. Figure 37 is a schematic drawing showing a connection point. Figure 38 shows a connection point and a webcam.

Each computer with a webcam connected to it can get the picture and process the image within less than 0.05 of a second. Therefore the data from the whole vision system is updated in less than 0.05 of a second.

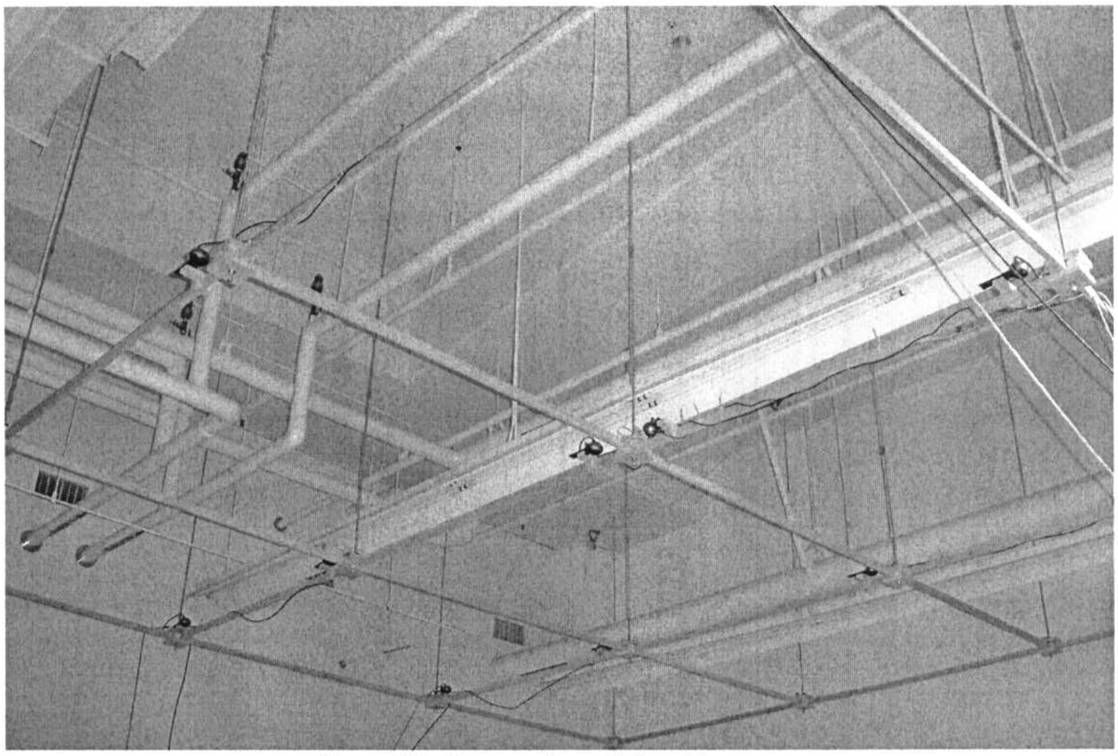


Figure 36- An overview of the vision array

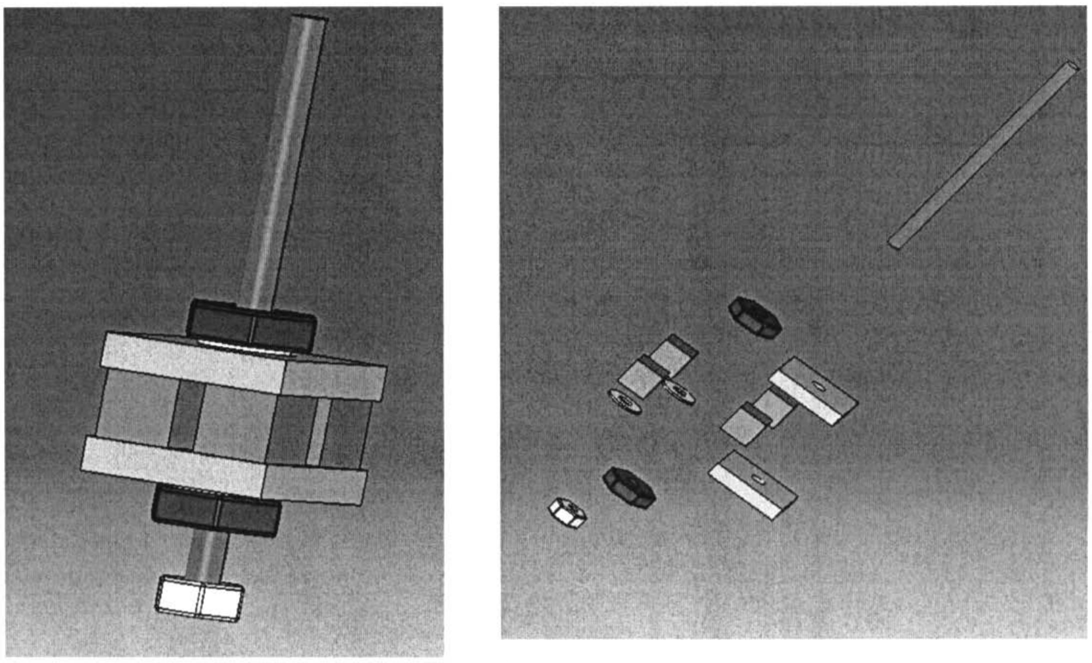


Figure 37- Schematic drawing of a connection point

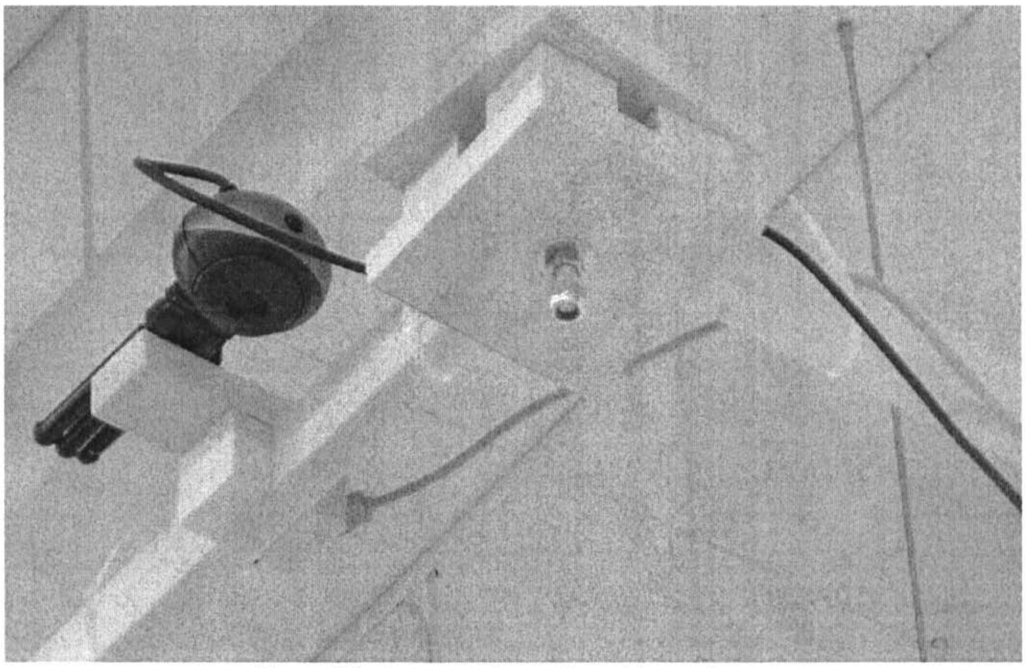


Figure 38- A connection point with a webcam

5.4.2 Open Loop Response

In this test the response of the hovercraft to some reference inputs is reported.

5.4.2.1 Forward Movement

In this case a same input signal applied to both motors ($u_1 = u_2$ in equation (28)). The results are shown in Figure 39 for different cases of the following input signals.
$u_1 = u_2 = 0.5,\ 0.7,\ 0.85,\ \mathit{and}\ 1.0$

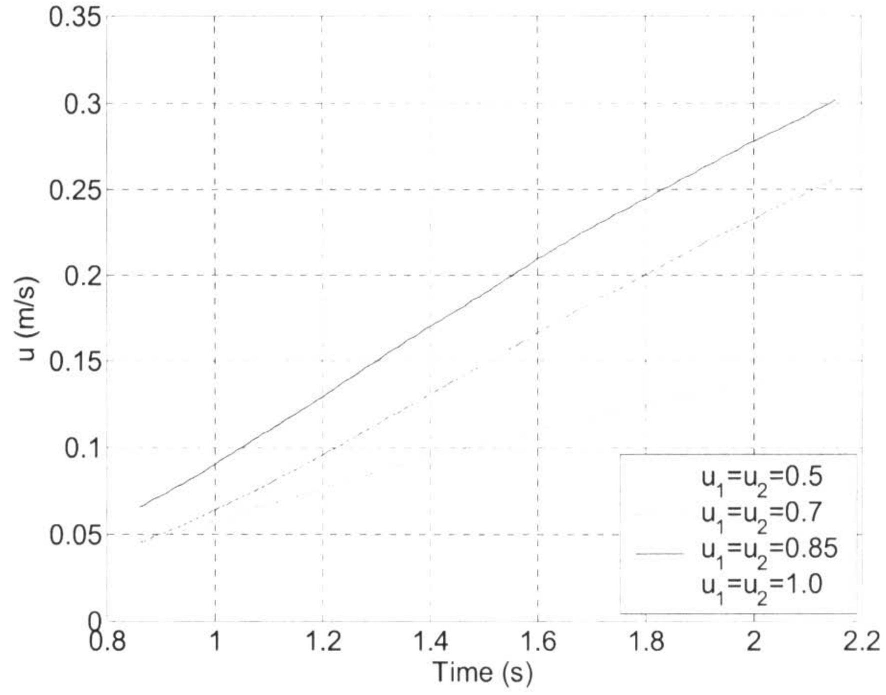


Figure 39- forward velocity (u) of hovercraft caused by different input signal values

The data used in Figure 39 are filtered as described in section 3.2.

5.4.2.2 Rotation

In this case a same input signal but in reverse direction applied to both motors ($u_1 = -u_2$ in equation (30)). The results are shown in Figure 39 for different cases of the following input signals.
$u_1 = -u_2 = 0.5,\ 0.7,\ 0.85,\ \mathit{and}\ 1.0$

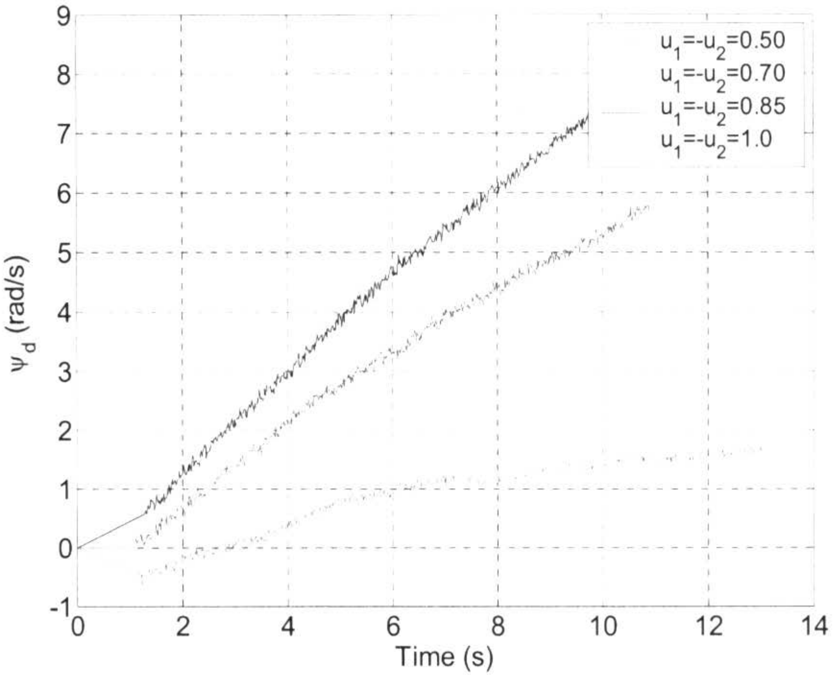


Figure 40- Angular velocity ($\dot{\psi}$) of hovercraft caused by different input signal values

In Figure 40 Wireless InertiaCube3 is used to get data, so the data are not filtered and they are angular velocities measured by sensor.

5.4.3 RHC of a Single Hovercraft

In order to get the results of one hovercraft controlled by RHC approach, a setup used which is shown in Figure 41. The RHC method used in this report is based on the method presented in [6], the sensor is wireless InertiaCube3 and the control parameters are as follows:

Nx = 2	Number of states (ψ and $\dot{\psi}$)
Nz = 1	Number of flat outputs (ψ)
Nu = 1	Number of inputs (u_1[27])
Ny = 2	Number of outputs (ψ and $\dot{\psi}$)
Nd = 2	Maximum number of derivatives from flat outputs
Ns = 6	Number of Spline control points
Ni = 51	Number of Spline interpolation points

[27] See equation (41) for clarity. Note that only one hovercraft is used at this step.

T = 3.0 (sec)	Prediction Horizon
delta = Ts =0.05 (sec)	Execution Horizon which is equal to the period of the periodic function (Ts) defined for real time execution
Nc = Ns	Number of varying control points
Imethod = 1	It means that the cubic spline is used for interpolation

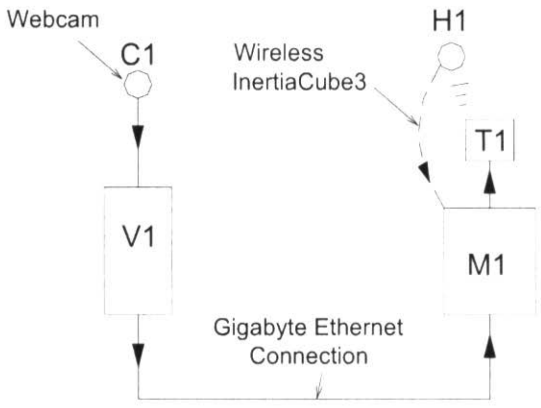


Figure 41- Schematic layout of experimental setup for one hovercraft and one camera[28]

For single hovercraft two groups of tests are performed only in rotation direction; in the first one, the vision system does not used and all of the data were measured by Wireless InertiaCube3 with the *partial compass mode*. The other is done were the combination of vision system and wireless InertiaCube3 was used.

5.4.3.1 Rotation Direction with wireless InertiaCube3

In this case two tests have been done, Regulation and Tracking.

Regulation with RHC

In this case, the hovercraft was started with a Yaw angle which was not equal to zero and with the RHC Regulating controller should go back to zero angle.
The result is shown in Figure 42.

[28] C1, V1, M1, T1, and H1 are described in the description of Figure 21.

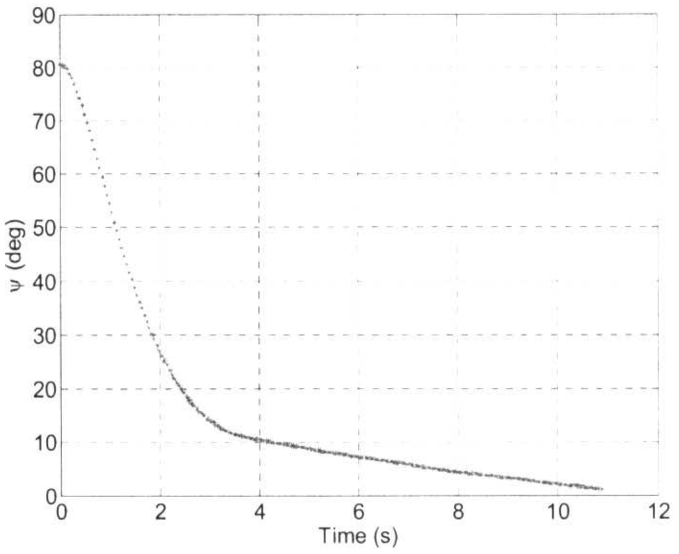


Figure 42- regulating Yaw angle of the hovercraft with RHC controller

As shown in Figure 42, the angle of hovercraft is getting back to zero by the RHC regulator.

Tracking with RHC

In this case, the Yaw angle of hovercraft should follow a trajectory. The trajectory was sinusoidal with respect to time. The result is shown in Figure 43.

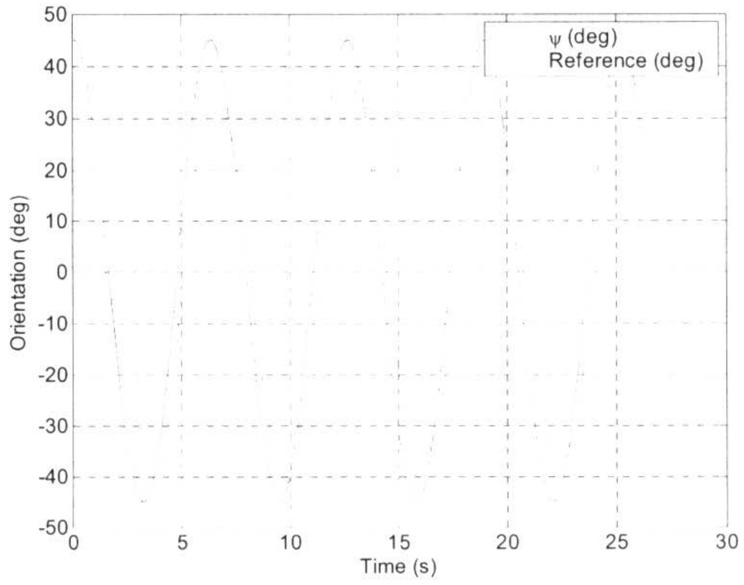


Figure 43- Yaw angle tracking of the hovercraft with RHC controller when only wireless Inertiacube3 is used

As shown in Figure 43, the angle of hovercraft (red line) is following the reference trajectory (blue line). As mentioned before the sensor is used in the *partial compass mode*, and it had drift, so it followed the trajectory but not very well in Figure 43. In addition, in reality the zero angle of hovercraft was changing because of the drift and it is not shown in the Figure.

In the next section the data from Vision System is added and the performance of the system is better.

5.4.3.2 Rotation Direction with wireless InertiaCube3 and Vision System

In Figure 44 the angular velocity ($\dot{\psi}$) is measured by wireless Inertiacube3 and the angle is measured using the vision system and the hovercraft is able to follow that better. The green line shows the ψ angle measured by wireless InertiaCube3 where the drift in the measurement is obvious.

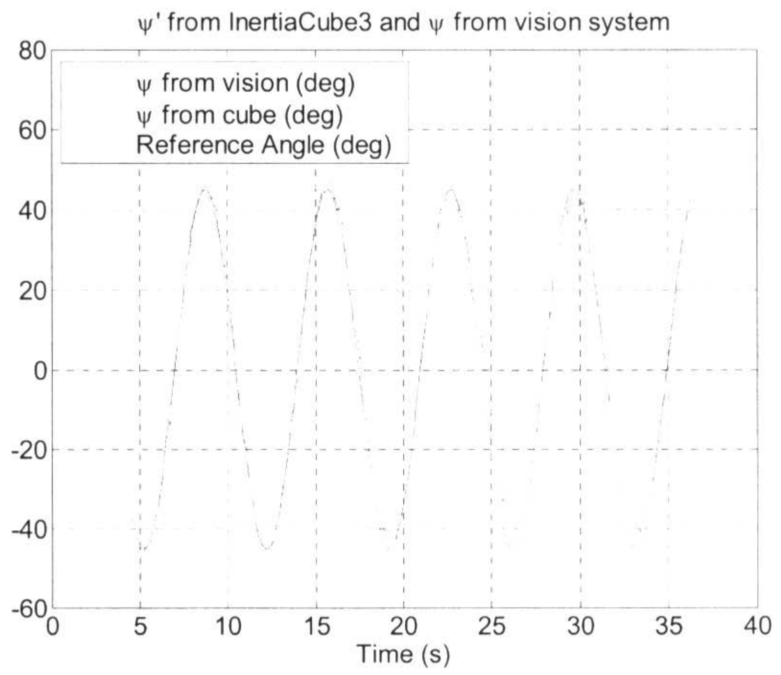


Figure 44- Yaw angle tracking of the hovercraft with RHC controller, when ψ is measured by vision system and $\dot{\psi}$ is measured by wireless InertiaCube3 (the green line shows the ψ angle measurement by InertiaCube3 where the drift in the measurement is obvious)

In the next test only vision system is used. So the angle and angular velocity of hovercraft measured and calculated by data from vision system. By using vision system, only the angle of hovercraft can be measured, and the angular velocity was calculated by finite difference method, and filtering the data. Consider equation (36) with *filter* value equals to 5. So the measured data in 5 steps before the current state were used to calculate angular velocity of current state. This filter value was obtained by trial and error. In this case with the real model parameters, the system was not working and only when the a_3

value in equation (41) was selected 4 times greater than the actual value, the system was stable and worked. The data are shown in Figure 45 and Figure 46.

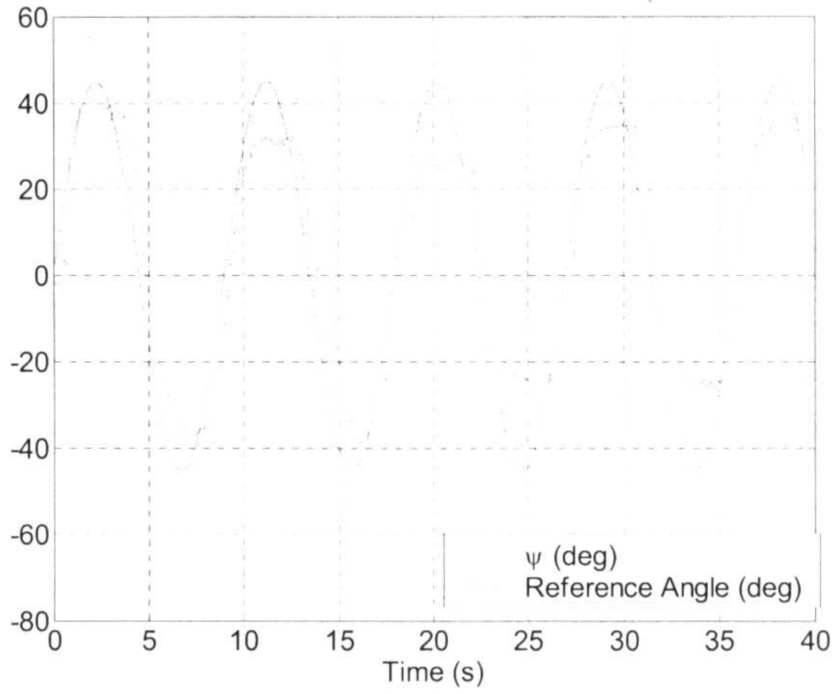


Figure 45- Yaw angle tracking of the hovercraft with RHC controller, when ψ is measured by vision system and $\dot{\psi}$ is calculated from the data of vision system.

As shown in the Figure 45 the tracking is not very good because the model used in prediction of states (the model used in controller) was not accurate which explained before.

Next Figure shows the angular velocity measured by InertiaCube3 and the angular velocity calculated from vision data. It can be concluded from Figure 46 that the calculated data from vision, has about 0.2 seconds delay from the values measured by wireless InertiaCube3. It is noticeable because in this test, which vision system is involved, two periodic functions were defined to do two time consuming jobs (reading vision data and calculating the outputs (RHC)). The period of the first task (reading vision data) is 0.03 seconds and the period of the second task is 0.05 seconds. Since the data from 5 steps before (0.15 seconds before) are used as well as the data in the current time, some part of this delay is expectable.

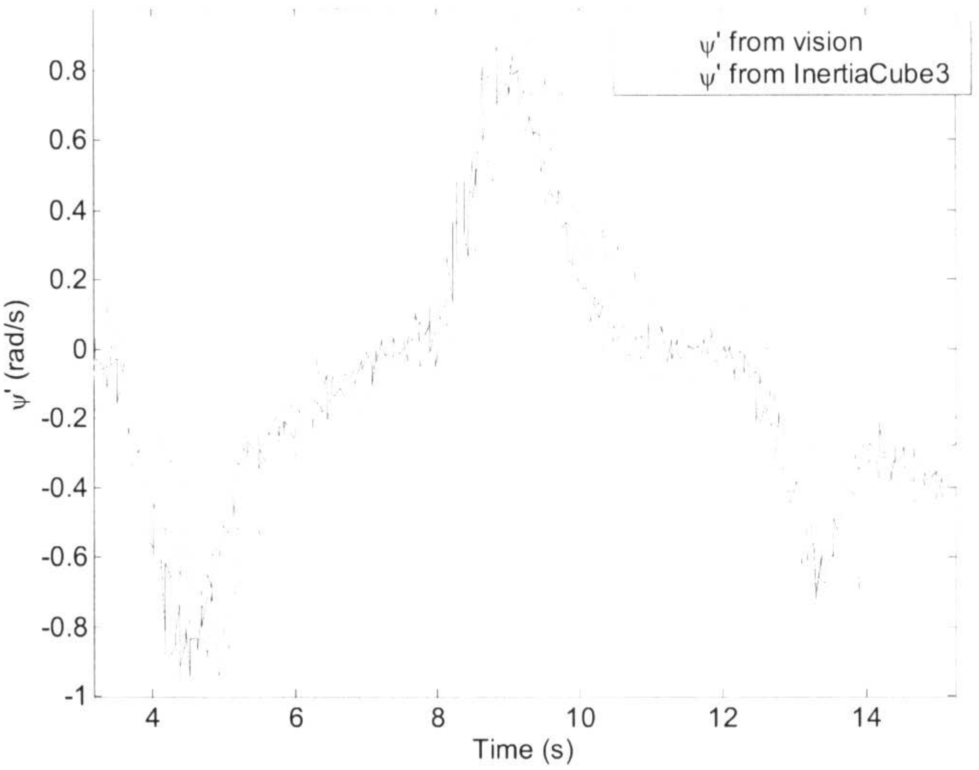


Figure 46- $\dot{\psi}$ measured by wireless InertiaCube3 (blue line) and $\dot{\psi}$ calculated from vision data (red line). A delay in the red line can be concluded comparing to the blue line.

5.4.4 RHC of two Hovercrafts

In order to get the results of two hovercrafts controlled by RHC approach, the experimental setup illustrated in Figure 47 is used. For simplicity only one camera is used, and two hovercrafts are controlled by one computer. One of the hovercrafts uses both data from vision and wireless InertiaCube3 (subsystem #1), and the other one uses only data from vision (subsystem #2).

Three periodic tasks are defined in this experiment. One for reading vision data, one for controller #1, and the other for controller #2. For solving RHC problem for both subsystems, RHCOOL is used and the control parameters are the same as controller for one system (data presented for controller in the beginning of section 4.3). The only difference is the execution horizon (or the period of the periodic task). The second subsystem, which has more uncertainty because of using vision data for angular velocity calculation, has the execution horizon of 0.05 seconds while for the first subsystem the execution horizon is 0.10 seconds. Only one model is used for the controllers of both subsystems. As discussed before, the model parameter a_3 in equation (41) considered to be about four times greater than the actual value because of the problem in angular velocity calculated by vision data.

In order to illustrate that the performance does not decreased when one computer is used for both hovercrafts, the result of tracking with the same model parameter when one computer controls only one hovercraft, is also presented.

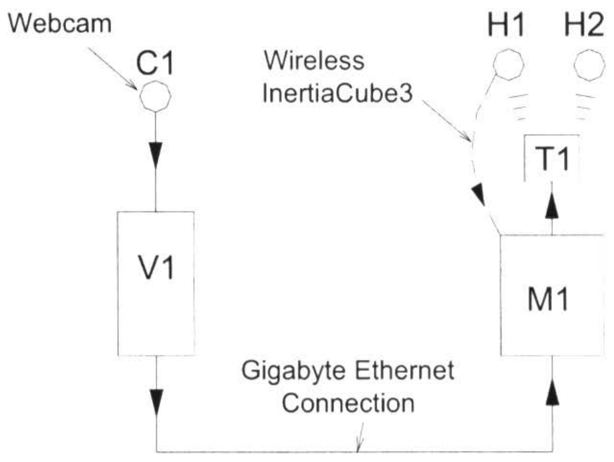


Figure 47- Schematic layout of experimental setup for two hovercrafts, one camera, and one wireless InertiaCube3 sensor[29]

In Figure 47 the computer used for controlling two hovercrafts (M1) is a Pentium 4, with one CPU of 2.8 GHz speed. This is the same as computer used in Figure 41 for one hovercraft.

As two vehicles are controlled with a single computer, the scheduling approach described in section 3.4 is used. The equations of motion of two hovercrafts are equations (41) with the following parameters:

First hovercraft (i=1):

$b'_{1,3} = -0.15$

$a_{1,3} = -1.7$

And the second hovercraft (i=2):

$b'_{2,3} = -0.11$

$a_{2,3} = -2.1$

But the parameters used in controller are as follows:

$b'_{1,3} = b'_{2,3} = 0$

$a_{1,3} = a_{2,3} = -7.5$

[29] C1, V1, M1, T1, H1, and H2 are described in the description of Figure 21.

As discussed above, the model parameters are changed because of the problem in calculated angular velocity of the second subsystem from vision data. Also only one model is used in the controller of both subsystems. The results are as follows:

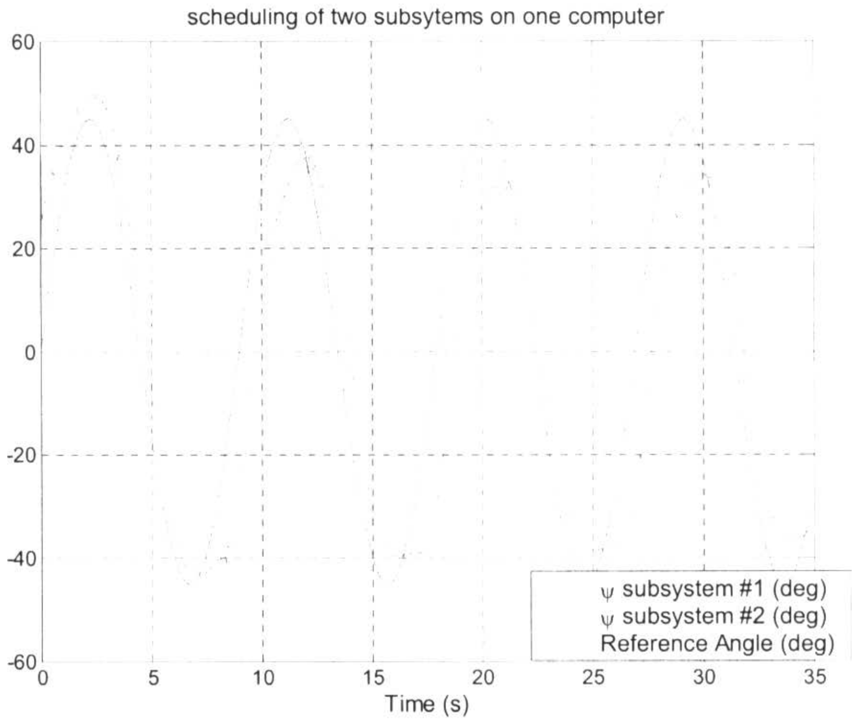


Figure 48- Yaw angle tracking of the first and second hovercrafts (red and green lines respectively) and the reference path (blue line) (after scheduling)

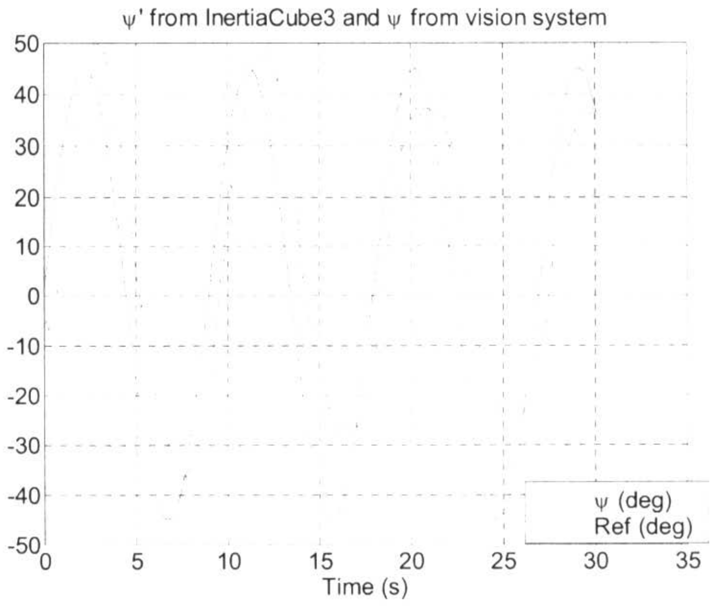


Figure 49- Yaw angle tracking of the first hovercraft (red line) and the reference path (blue line) – only first hovercraft with the same computer and model parameters (before scheduling)

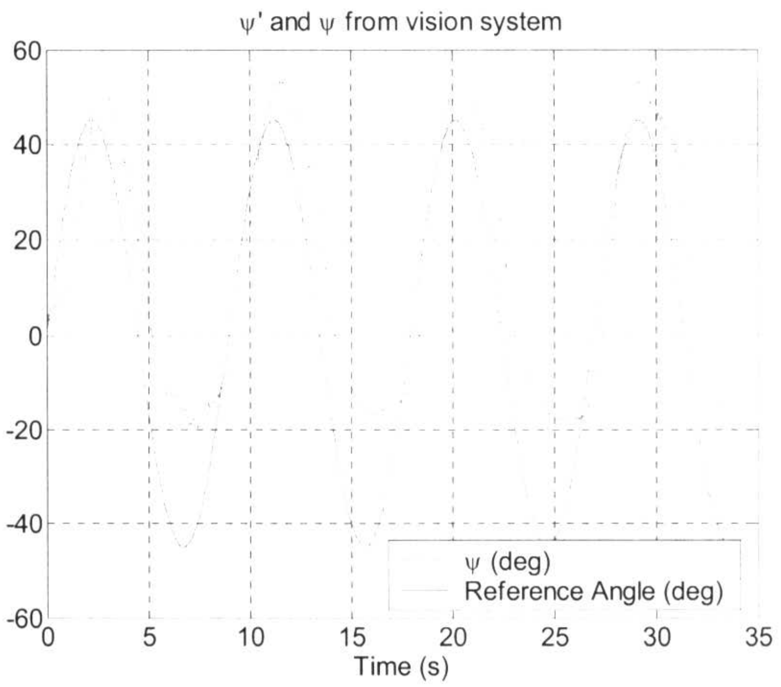


Figure 50- Yaw angle tracking of the second hovercraft (red line) and the reference path (blue line) – only second hovercraft with the same computer and model parameters (before scheduling)

Discussion

By comparison of Figures 48 to 50, the performance of the system when scheduling is performed does not change. Since the model parameters are not accurate, the performance of both subsystems is not very good even when they are controlled individually (only one hovercraft in one computer). However in Figure 44 the hovercraft was controlled very well, individually and followed the same path, perfectly. Because it was using wireless InertiaCube3 for measuring angular velocity and the model parameters were accurate. Since only one wireless InertiaCube3 was available, the scheduling of two hovercrafts was done in the foresaid manner (one with wireless InertiaCube3 and the other with vision data for angular velocities). Therefore the model parameters were changed (in order to be able to work with angular velocities calculated from vision data) and the tracking was not very good. Some videos are prepared showing the scheduling of two subsystems with one computer where following the defined path (only yaw angle is following). Video1, Video2, and Video3.

In all of the following clips, the reference path is a sinusoidal wave versus time with the amplitude of 45 degrees.

$$reference\ path = 45 \times \sin(t\omega)$$

In deferent clips the frequency of this wave (ω) was changed.

Video1:

w = 0.3	The frequency of the reference path
Ts1 = 0.15	The period of the first subsystem
Ts2 = 0.05	The period of the second subsystem
Ts3 = 0.03	The period of the reading data

Video2:

w = 0.5	The frequency of the reference path
Ts1 = 0.15	The period of the first subsystem
Ts2 = 0.05	The period of the second subsystem
Ts3 = 0.03	The period of the reading data

Video3:

w = 0.7	The frequency of the reference path
Ts1 = 0.15	The period of the first subsystem
Ts2 = 0.05	The period of the second subsystem
Ts3 = 0.03	The period of the reading data

5.5 Conclusions and Future Work

5.5.1 Conclusions

In this report, a systematic approach is developed for scheduling computation and determination of the execution horizon for multiple uncertain RHC systems. It was shown that using a rate monotonic priority assignment method combined with analytical bounds on the prediction error, the problem of scheduling multiple uncertain plants can be cast into an appropriate constrained optimization problem. The constraints guarantee that the processes will be schedulable, and the optimization provides performance robustness to uncertainty. The proposed method was first applied via Matlab to a double integrator example problem demonstrating the validity of the approach. Performance of the method was also demonstrated by applying it to some sets of under actuated miniature hovercrafts in a hard real time environment. The latter was done by developing a RHC Object Oriented Library (RHCOOL). This report was explaining Task #7 and Task #8 of the first proposal.

5.5.2 Future Work

1- Design a scalable, dynamic run-time scheduling algorithm for multiple RHC running on distributed digital processors (**Task 11**)

Dynamic scheduling of computational and communication resources for multiple RHC control problems is very important since scheduling requirements and resulting performance can vary dramatically as the system dynamically varies over time. This can be achieved by formulating the uncertainty bounds as a function of state, time, external inputs, and other factors that influence the local properties of these bounds. These time varying bounds can then be optimized on-line as the system is varying.

Finally, the optimization domain will be expanded to include a multi-cluster framework (see Figure 51) which will allow the number of processors, inter-processor communication, and UAV communication to be dynamically optimized. This will lead towards a new generalization of scheduling tradeoffs between performance and resources using control metrics. This could include dynamic varying and optimizing:

- The number of computing nodes per RHC subsystem
- Computational delays
- Communication delays
- The RHC horizons
- Additional RHC control design parameters
- Computation for centralized cooperative control schemes
- Computation and communication for low level conl

2- RHC and scheduling algorithms on multi-rotorcraft experimental test-bed (**Task 13**)

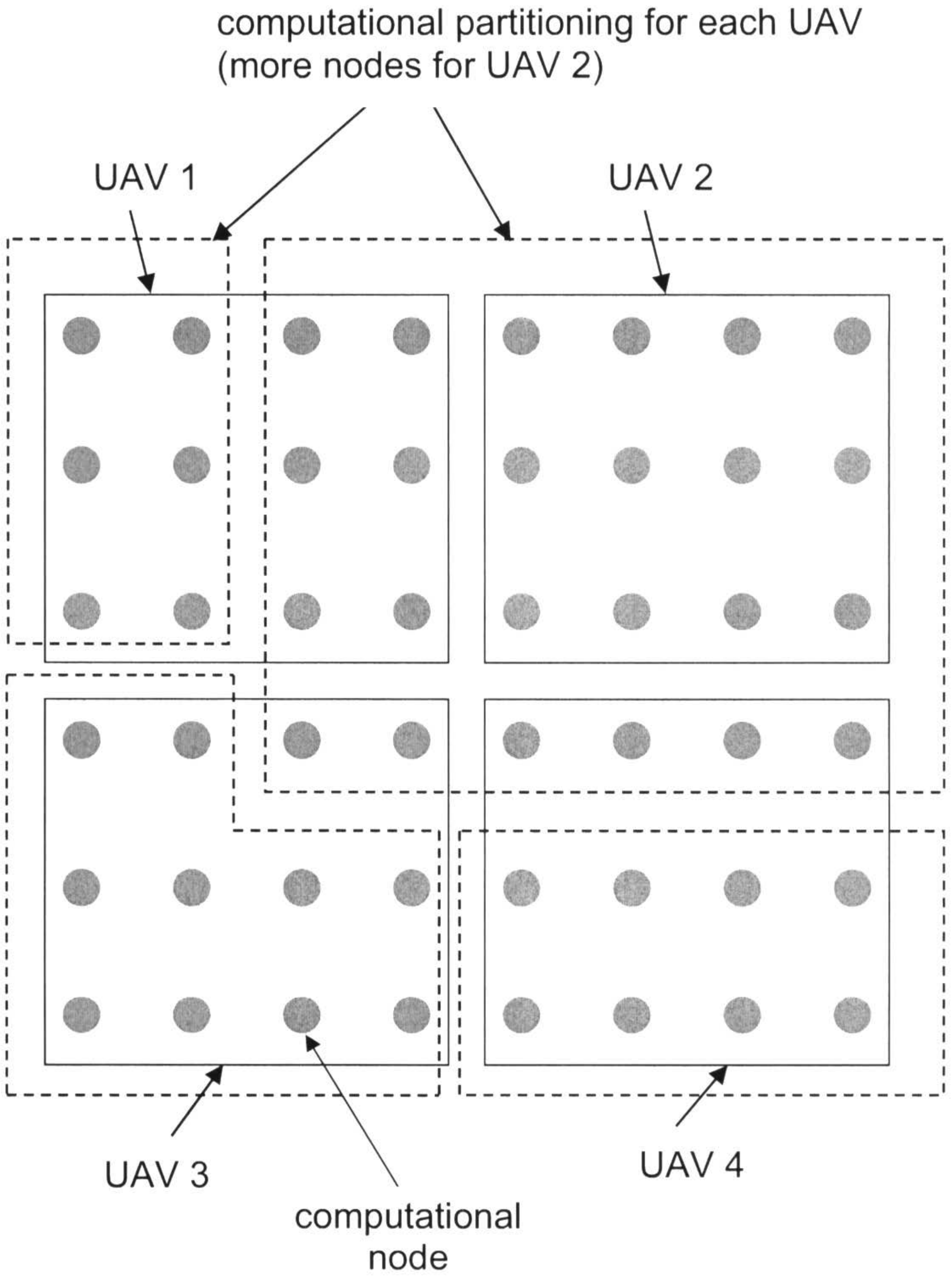


Figure 51- Multi-cluster framework for multi-UAV control systems at the CIS lab
(48 node example)

5.6 References

[1] B. Gholami, B. W. Gordon, C. A. Rabbath, "Real-time Scheduling of Multiple Uncertain Receding Horizon Control Systems", 2005 (to be submitted to *AIAA Journal of Aerospace Computing, Information, and Communication*).

[2] Cutler, C. R., Ramaker, B. L., 1980, "Dynamic matrix control – A computer control algorithm", *Proceedings of Joint Automatic Control Conference*, San Francisco, CA, Paper WP5-B.

[3] Mayne, D. Q., Rawlings, J. B., Rao, C. V., Scokaert, P. O. M., 2000, "Constrained model predictive control: Stability and optimality", *Automatica*, 36, pp. 789-814.

[4] Cannon, M., Kouvaritakis, B., Rossiter, J. A., 2001, "Efficient active set optimization in triple mode MPC", *IEEE Transactions on Automatic Control*, 46 (8), pp. 1307-1312.

[5] Milam, M. B., Mushambi, K., Murray, R. M., 2000, "A new computational approach to real-time trajectory generation for constrained mechanical systems", *Proceedings of the 39th IEEE Conference on Decision and Control*, Sydney, Australia, pp. 845 – 851.

[6] M. B. Milam, R. Franz, J. E. Hauser, R. M. Murray, "Receding horizon control of a vectored thrust flight experiment", *submitted to IEE Proceedings on Control Theory and Applications*.

[7] D. Henriksson, J. Akesson, 2004, "Flexible implementation of Model Predictive Control using sub-optimal solutions", *Technical Report, Department of Automatic Control, Lund Institute of Technology*, Lund, Sweden.

[8] H. Kopetz, "Real-time systems: Design principles for distributed embedded applications", Chapters 9 and 11, *Springer*, 1997.

[9] C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of ACM*, 20 (1), 1973.

[10] D. Henriksson, "Flexible scheduling methods and tools for real-time control systems", *Licentiate Thesis*, December 2003. Department of Automatic Control, Lund Institute of Technology, Sweden.

[11] Chen, H., Allgower, F., 1998, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability", *Automatica*, 34, pp.1205-1217.

[12] Jadbabaie, A., 2000, "Receding horizon control of nonlinear systems: A control Lyapunov function approach", *Ph.D Thesis, California Institute of Technology*, Pasadena, CA.

[13] B. Gholami, B. W. Gordon, C. A. Rabbath, "Real-time scheduling of multiple uncertain receding horizon control systems", *Technical Report 07-05*, Control and Information Systems Laboratory, Concordia University, Montreal, 2005.

[14] H. Khalil, *Nonlinear Systems*, Prentice Hall, 2002.

[15] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems", *PhD Thesis, California Institute of Technology,* Pasadena, CA, 2003.

[16] B. J. Young, R. W. Beard, J. M. Kelsey, "A control scheme for improving multi-vehicle formation maneuvers", *American Control Conference*, Arlington, VA,

pp. 704-709, 2001.

[17] Desai, J. P., Ostrowski, J., Kumar, V., 1998, "Controlling formations of multiple mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp. 2864-2869.

[18] Pomet, J.-B., Thuliot, B., Bastin, G., Campion, G., 1992, "A hybrid strategy for the feedback stabilization of nonholonomic mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, pp. 129 – 134.

[19] Feddema, J. T., Lewis, C., Schoenwald, D. A., 2002, "Decentralized control of cooperative robotic vehicles: theory and application", *IEEE Transaction on Robotics and Automation*, 18 (5), pp. 852-864.

[20] J.-P. Aubin, A. Cellina, *Differential inclusions: set-valued maps and viability theory*, Springer-Verlag, 1984.

[21] B. Gholami, B. W. Gordon, C. A. Rabbath, "Uncertain Nonlinear Receding Horizon Control Systems Subject to Non-Zero Computation Time", *44th IEEE Conference on Decision and Control and European Control Conference ECC*, Seville, Spain, 2005.

[22] A. P. Aguiar, L. Cremean, and J. P. Hespanha, "Position Tracking for a Nonlinear Underactuated Hovercraft: Controller Design and Experimental Results", CDC, 2003.

[23] pciBIRD™, Technical Reference Guide, ASCENSION TECHNOLOGY CORPORATION, 2002

[24] Wireless InertiaCube3 Supplemental Manual, Doc. No. 072-00096-0E05, InterSense, Inc, 2005.

[25] InertiaCube3 Manual, Doc. No. 072-00094-0D05, InterSense, Inc, 2005.

[26] J. L. Meriam, L. G. Kraige, "Engineering Mechanics Dynamics", Wiley, Fourth edition, 1998

[27] V. Gavrilets, B. Mettler, E. Feron, "Dynamic Model for a Miniature Aerobatic Helicopter", Technical Report, MIT.

[28] X. Ren, B. W. Gordon, "Helicopter control system and 3D motion tracking", Technical report, CIS Lab, 12/2003

[29] "Numerical Recipes in C", chapter 3.3 "Cubic Spline Interpolation", ISBN 0-521-43108-5.

[30] C. W. Mercer, "An Introduction to Real-Time Operating Systems: Scheduling Theory", School of Computer Science, Carnegie Mellon University, 1992.

[31] http://www.superdroidrobots.com/shop/item.asp?itemid=601

[32] http://www.amazon.com/exec/obidos/tg/detail/-/B0006TQ61M/qid=1143092249/br=1-14/ref=br_lf_t_14//104-4082232-4929547?v=glance&s=imaginarium&n=13685221

[33] http://www.rcuniverse.com/product_guide/servoprofile.cfm?servo_id=79

[34] http://www.intersense.com/support/faqs/ic2.htm

[35] http://www.sce.carleton.ca/courses/sysc-4102/hard-real-time.pdf

[36] http://www.nag.co.uk/numeric/cl/nagdoc_cl08/pdf/E04/e04ucc.pdf

[37] http://www.nag.co.uk/numeric/cl/nagdoc_cl08/pdf/E04/e04xzc.pdf

[38] http://www.sbsi-sol-optimize.com/manuals/SNOPT%20Manual.pdf

[39] http://www.cs.bu.edu/~best/crs/cs835/S96/lectures/scheduling.html

[40] G. C. Buttazzo, "Rate Monotonic vs. EDF: Judgment Day", *Real-Time Systems*, 29, 5–26, 2005.
[41] http://feanor.sssup.it/~giorgio/slides/rts/HB.pdf

5.7 APPENDIX A: Wireless Inertiacube3 troubleshooting

In order to start working with wireless Inertiacube3, the following steps should be done:

1. Attach the Receiver to the USB port of computer and install the driver and programs as described in [24].

2. Attach a 6 to 9 V battery to the sensor[30] and run "Device Tool". Click on the yellow bulb on top left of the screen of Device Tool. If it could not detect the sensor, from the "Wireless" menu, run "search for stations". If it could not detect the sensor again, restart the sensor (just detach the battery and attach it again) and click on the yellow bulb. If the problem exists do the previous command (restarting the sensor and clicking on the yellow bulb) for 2 to 3 times. If the sensor does not work, consult with the manufacturer.

3. When the sensor was detected, run ISDemo (with dll compatible mode) to see how the sensor is working.

For trouble shooting refer to [25]. The most popular problems and probable ways to fix them are listed bellow:

Problem	Reason/Solution
The sensor interference or lack of accuracy	Check for the following if the sensor is in full or partial compass mode: 1) Placement directly on top of metal. You should place the wireless InertiaCube3 an inch or two away. If the InertiaCube3 must be mounted in a metallic environment, follow the steps in [25] (Section 4.7) to perform a magnetic calibration using the "Compass Calibration Tool". 2) A varying magnetic field may exist in the area. Check the area with a compass to be sure about the existence of the varying magnetic field. In this case the best thing is changing the working area.
Orientation is drifting uncontrollably	1) Make sure that the sensor is properly plugged in. 2) Keep the sensor still for 10 seconds after connecting with ISDEMO or DLL. 3) Compass is turned off in ISDEMO.

The user is encouraged to see FAQ [34].

[30] For specifications of battery see [24].

5.8 APPENDIX B: RHC.h

```
class RHC {

        // internal variables for Powell's optimization functions
        int ncom;
        double *pcom,*xicom;
        dVector *ptt_RHC, *pt_RHC, *xit_RHC, *xt_RHC;

public:

        double t0; // initial time
        double t; // current time

        double tb;      // time that interpolation begins

        dVector *ti; // interpolation time
        dVector *ts; // time at spline points

        double dti; // interpolation time interval

        double T; // RHC optimization horizon
        double delta; // RHC execution horizon

        int Nx; // number of states
        int Nu; // number of inputs
        int Nz; // number of flat outputs
        int Ny; // number of outputs

        dVector *X0; // initial condition state
        dVector *U0; // initial input

        dVector *X; // current state (at time t)
        dVector *U; // current input
        dVector *Y; // current output
        dVector *Z; // current flat output

        dVector *Xd; // current state derivative

        int Ns; // number of spline control points for each output
        int Ni; // number of spline interpolation points for each output
        int Nc; // number of varying control points
        int Nd; // maximum number of output derivatives required

        int ks; // order of bspline

        dMatrix *Cs; // spline control points Cs[q][j] for flat output q, control point j
        dVector *Cv; // vector containing all varying control points

        dMatrix *Xi; // state Xi[j][i] for each interpolation point i
        dMatrix *Ui; // input Ui[j][i] for each interpolation point i
        dMatrix *Yi; // output Yi[j][i] for each interpolation point i

        // flat output q (1 to Nz) for each derivative id (0 to Nd)
        // at each interpolation time i
        dMatrix_array *Zi; // Zi[q][id][i]

        // flat output value at the beginning of the interpolation time t0
        dMatrix *Zi0; // Zi0[q][id]

        // flat output value at the end of the interpolation time t0+T
        dMatrix *ZiT; // ZiT[q][id]

        // cost function parameter matrices
        dMatrix *Q; // (Nx x Nx)
        dMatrix *R; // (Nu x Nu)
        dMatrix *P; // (Nx x Nx)
        // Bs - bspline basis functions, Bs[id][i][j] for derivative id (0 to Nd),
        // interpolation time i (1 to Ni), control point j (1 to Ns)
        dMatrix_array *Bs; // Bs[id][i][j]

        // second derivative at control points C2[q][j] for flat output q,
        // control point j for cubic spline interpolation method
        dMatrix *C2;

        dMatrix *vi;

        // output variables
        double tout_final; // final output time
        int Nout; // number of output points
        int Nsamp; // number of output samples
        dMatrix *Xout; // state Xout[j][i] for each time tout[i]
        dMatrix *Uout; // input Uout[j][i] for each time tout[i]
        dMatrix *Yout; // output Yout[j][i] for each time tout[i]
        dMatrix *Zout; // flat output Zout[j][i] for each time tout[i]
        dVector *tout; // output time with spacing at the interpolation points dti

        int Imethod; // interpolation method [ 1 = cubic spline, 2 = bspline ]

        // contructor function
        RHC(double t0, double T, double delta, int Nx, int Nu, int Nz, int Ny,
                dVector &x0, dVector &u0, int Ns, int Ni, int Nc, int Nd,
                double tout_final, int Imethod, int ks);

        // destructor function
        ~RHC();

        // function to optimize using Powell's method
        double func_powell(dVector &c);

        // cost function for trajectory generation
        double calc_J(dVector &c);

        // optimize the cost function
        optimize_cost();

        interpolate();

        // calculate Xi and Ui based on the flat outputs Zi
        calculate_XU();

        // calculate the input at time tc
        calculate_U(double tc);

        // simulate the system from the current time t to the final time tf
        // using the optimal input Ui and a time step dt
        // the result is stored in the output variables Xout, Uout, Yout, tout
        simulate_interval(double tf, double dt);

        // calculate the derivative vector xd = f(x,u,t) of the actual system
        calculate_xd(dVector *x, dVector *u, double t, dVector *xd);

        calculate_outputs(dVector *x, dVector *u, double t, dVector *y, dVector *z);

        // save variables into a text file
        save_data(char *file_name);

        // save flat output variables over the current
        // interpolation interval into a text file
        save_flat_outputs(char *file_name);

        RHC::compute_flat_ICs(dVector &x0, dVector &u0, dVector &ud0);

        u_compute_C4(double *t, double *x, double *xd, double *xd2, double *xd3, double
*xd4, double *y, double *yd, double *yd2, double *yd3, double *yd4, double *u1, double
*u2, double *psi, double *psid, int ni);

        // Powell's method optimization functions
        void powell(dVector &c, double **vi, int n, double ftol, int &iter, double &fret);
        void linmin(double p[], double xi[], int n, double *fret);
        void mnbrak(double *ax, double *bx, double *cx, double *fa, double *fb, double
*fc);
        double brent(double ax, double bx, double cx, double tol, double *xmin);
        double f1dim(double x);

};
```

5.9 APPENDIX C: Program_heli_local.cpp

```
#include <cmath>   // math functions
#include <cstdio>  // standard I/O functions
#include <cstring> // string manipulation functions

#include <iostream>  // console stream I/O
#include <fstream>   // file stream I/O
#include <strstream> // string stream I/O

#include <conio.h> // console I/O functions such as getch()

using namespace std;

#include "nrutil.h"
#include "recipes.h"
#include "cubic_spline.h"
#include "timer.h"

#include "dVector.h"
#include "dMatrix.h"
#include "dMatrix_array.h"

#include "RHC.h"

using namespace std;

void main()
{
        double t0,T,delta,tf,t;
        int Nx, Nu, Nz, Ny;
        int Ns, Ni, Nc, Nd;
        int Imethod;
        int ks;

        t0 = 0.0;

        // setting key dimensions, refer to RHC.h
        Nx = 6;
        Nu = 2;
        Nz = 2;
        Ny = 3;
        Nd = 2;

        dVector x0(1,Nx), u0(1,Nu);

        Ns = 3+1;
        Ni = 50+1;

        T = 1.0;

        delta = 0.1;

        // number of free optimization parameters
        // *** this one needs to be carefully explained
        Nc = Nz*Ns + 1;

        tf = 2*3.14159;

        Imethod = 1; // cubic spline interpolation method

        // bspline order
        ks = 6; // must be at least nd + 1

        // declare two RHC objects
        RHC rhc1(t0,T,delta,Nx,Nu,Nz,Ny,x0,u0,Ns,Ni,Nc,Nd,tf,Imethod,ks);

        // set ICs
        rhc1.X->e(1) = 0.0; // u
        rhc1.X->e(2)  = 0.0; // v
        rhc1.X->e(3)  = 0.0; // r
        rhc1.X->e(4)  = 0.0; // psi
        rhc1.X->e(5)  = 2.0; // xc
        rhc1.X->e(6)  = 0.0; // yc

        // optimize the cost function over one horizon
        rhc1.optimize_cost();

        // save the flat outputs and related states for this horizon
        // the main purpose of this is for debugging and
        // understanding how good one optimization horizon is
        // this may be good for tuning the parameters
        // also good for checking your flat output
        // description of your system
        rhc1.save_flat_outputs("f.dat");

        // the first two calls are not needed to simulate
        // this RHC example

/*
                // interpolation time
                fprintf(fp,"%le ",ti->e(i));

                // interpolation states
                for(j=1;j<=Nx;j++) {
                        // input Xi[j][i]
                        fprintf(fp,"%le ",Xi->e(j,i));
                }

                // inputs corresponding to flat outputs
                for(j=1;j<=Nu;j++) {
                        // input Ui[j][i]
                        fprintf(fp,"%le ",Ui->e(j,i));
                }

                // flat outputs
                for(j=1;j<=Nz;j++) {

                        for(id=0;id<=Nd;id++) {
                                // Zi[q][id][i]
                                fprintf(fp,"%le ",Zi->e(j,id,i));
                        }
                }

*/

        // simulate RHC controller for 3dof helicopter in local coordinates
        t = delta; // delta = execution horizon (the sampling rate)
        while(t < (tf+1e-7)) {

                rhc1.optimize_cost(); // optimize RHC over T

                // simulate system until time t (t <= T)
                // in this case since we are incrementing t
                // by delta (the execution horizon),
                // so we simulate over the execution horizon
                // this approximates the behavior of a real RHC system
                // when we apply the input over the execution horizon.
                // the user can specify the actual model that is
                // used for the simulation in RHC_user (calculate_xd).
                // this can be useful to test the robustness to uncertainty
                // the dt for the simulation is made smaller than dti (T/(Ni-1))
                // to get a more accurate simulation using Euler's method
                rhc1.simulate_interval(t,rhc1.dti/1000);

                t += delta;
        }

        // save the simulation output into a file
        // this function saves the output data at time intervals
        // of dti so there isn't too much data
        rhc1.save_data("a.dat");
/*
        for(i=1;i<=Nsamp;i++) {
                fprintf(fp,"%le ",tout->e(i));
                for(j=1;j<=Nx;j++) fprintf(fp,"%le ",Xout->e(j,i));
                for(j=1;j<=Nu;j++) fprintf(fp,"%le ",Uout->e(j,i));
                for(j=1;j<=Nz;j++) fprintf(fp,"%le ",Zout->e(j,i));
                for(j=1;j<=Ny;j++) fprintf(fp,"%le ",Yout->e(j,i));
                fprintf(fp,"\n");
        }
*/

        printf("\n\ndone.");
}
```

5.10 APPENDIX D: RHC_user_heli_local.cpp

```
#include <cmath>    // math functions
#include <cstdio>   // standard I/O functions
#include <cstring> // string manipulation functions

#include <iostream>  // console stream I/O
#include <fstream>   // file stream I/O
#include <strstream> // string stream I/O

#include <conio.h> // console I/O functions such as getch()

using namespace std;

#include "nrutil.h"
#include "recipes.h"
#include "cubic_spline.h"
#include "timer.h"

#include "dVector.h"
#include "dMatrix.h"
#include "dMatrix_array.h"

#include "RHC.h"

#include "heli_models.h"

RHC::calculate_U(double tc)
// calculate the input at time tc
{
        int i,j,i2;
        double u1,u2,t1,u;
        const double TINY = 1.0e-7;

        // approximate method using linear interpolation
        // between interpolation points ti
        i = (int)(tc/dti + TINY) + 1;
        for(j=1;j<=Nu;j++) {
                u1 = Ui->e(j,i);
                i2 = i + 1;
                if (i2 > Ni) i2 = Ni;
                u2 = Ui->e(j,i2);
                t1 = (i - 1)*dti;
                u  = u1 + (u2-u1)/dti*(tc - t1);
                U->e(j) = u;
        }
}


RHC::calculate_xd(dVector *x, dVector *u, double t, dVector *xd)
{
        // use local 3dof heli model
        f_heli_3dof2(x->A,u->A,xd->A);
}


RHC::calculate_outputs(dVector *x, dVector *u, double t, dVector *y, dVector *z)
{
        // u = X[1]
        // v = X[2]
        // r = X[3]
        // psi = X[4]
        // xc = X[5]
        // yc = X[6]

        // y1 = xc, y2 = yc, y3 = psi
        y->e(1) = x->e(5);
        y->e(2) = x->e(6);
        y->e(3) = x->e(4);
        // z1 = u, z2 = psi
        z->e(1) = x->e(1);
        z->e(2) = x->e(4);
}

const int NMAX = 5000;
double xcd[NMAX],ycd[NMAX],v[NMAX],vd[NMAX];

RHC::calculate_XU()
// calculate Xi and Ui based on the flat outputs Zi
{
        int i,q;
        double *ti;
        double *u,*ud;
        double *u1,*u2,*psi,*psid,*psidd;
        double c_psi,s_psi,dt,V_inf;
        double S_fus_x,S_fus_y,m,Izz,l_tr,k_mr_x,rho_a,cx,cy;
        double sum1,sum2;

        // set the helecoper paramters with the values from the report
        S_fus_x = 0.1;
        S_fus_y = 0.22;
        m = 8.2;
        Izz = 0.28;
        l_tr = 0.91;
        k_mr_x = -0.3;
        rho_a = 1.0;
        cx = -0.5*rho_a*S_fus_x;
//      cy = -0.5*rho_a*S_fus_y;
        cy = 0.0;

        // this part is not so profound
        // is because we have two names for ti
        // the one inside the RHC class is actually a dVector pointer
        // this "ti" is a 1D array of doubles
        ti = this->ti->A;

        q   = 1;
        u   = Zi->M[q]->A[0];
        ud  = Zi->M[q]->A[1];

        q   = 2;
        psi   = Zi->M[q]->A[0];
        psid  = Zi->M[q]->A[1];
        psidd = Zi->M[q]->A[2];

        u1 = Ui->A[1];
        u2 = Ui->A[2];

        // compute vd[i], assuming cy = 0
        for(i=1;i<=Ni;i++) {
                // vd = -u*r + cy * v * V_inf / m;
                vd[i] = -u[i]*psid[i];
        }

        // compute v[i] using trapezoidal integration
        i = 1;
        sum1 = v[i] = Xi->e(2,i) = X->e(2); // ICs
        dt = T/(Ni-1);
        for(i=2;i<=Ni;i++) {
                sum1 += 0.5*( vd[i-1] + vd[i] )*dt;
                v[i] = Xi->e(2,i) = sum1;
        }

        // compute Ui
        for(i=1;i<=Ni;i++) {

                V_inf = sqrt(u[i]*u[i] + v[i]*v[i]);

                //      ud = v*r - k_mr_x * Tmr / m + cx * u * V_inf / m;
                u1[i] = ( ud[i] - v[i]*psid[i] - cx * u[i] * V_inf / m ) / ( -k_mr_x / m
);

                // rd = -Ttr*l_tr/Izz;
                u2[i] = psidd[i]/(-l_tr/Izz);
        }

        // calculate Xi
        for(i=1;i<=Ni;i++) {
                Xi->e(1,i) = u[i]; // u = X[1]
                Xi->e(2,i) = v[i]; // v = X[2]
                Xi->e(3,i) = psid[i]; // r = X[3]
                Xi->e(4,i) = psi[i]; // psi = X[4]
        }

        // need to integrate u, v to get xc, yc
        for(i=1;i<=Ni;i++) {
                c_psi = cos(psi[i]); s_psi = sin(psi[i]);

                // express velocity vl=[u,v] in global coord using Vg = R*vl
                xcd[i] = c_psi*u[i] - s_psi*v[i];
                ycd[i] = s_psi*u[i] + c_psi*v[i];
        }

        // trapezoidal integration
        i = 1;
        sum1 = Xi->e(5,i) = X->e(5); // ICs
        sum2 = Xi->e(6,i) = X->e(6);
        dt = T/(Ni-1);
        for(i=2;i<=Ni;i++) {
                sum1 += 0.5*( xcd[i-1] + xcd[i] )*dt;
                Xi->e(5,i) = sum1;
                sum2 += 0.5*( ycd[i-1] + ycd[i] )*dt;
                Xi->e(6,i) = sum2;
        }

}


double RHC::calc_J(dVector &c)
// cost function for trajectory generation
// c is very important -- all free control points
{
        int i,j,q,id,ic;
        double t0,tf,dt;
        double sum;
        double J,L;
        double x1,x2,x3,x4,x5,x6,u1,u2;
        double w,Ax,Ay,xr,yr,tr;

        w = 1.0;
        Ax = 2.0;
        Ay = 1.0;

        // the first control points are equal to the ICs
        j = 1;

        q = 1;
        Cs->e(q,j) = X->e(1); // u

        q = 2;
        Cs->e(q,j) = X->e(4); // psi

        ic = 0; // index for optimization vector c

        // this following part can vary depending
        // on how we want to arrange our free control points
        // of our flat outputs into the vector c
        // c = [u_free psi_free]

        // flat output #1 (u)
        q = 1;

        // get control points from optimization vector c
        for(j=2;j<=Ns;j++) {
                ic++;
                Cs->e(q,j) = c.e(ic);
        }

        // define boundary conditions of the spline
        id = 1;
        ic++;

        // Zi0 - boundary condition derivatives (including zero derivative) at t=0
        Zi0->e(q,id) = c.e(ic); // end point derivative is free

        ic++;

        // ZiT - boundary condition derivatives (including zero derivative) at t=T
        ZiT->e(q,id) = c.e(ic); // end point derivative is free

        // flat output #2 (psi)
        q = 2;

        // get control points from optimization vector c
        for(j=2;j<=Ns;j++) {
                ic++;
                Cs->e(q,j) = c.e(ic);
        }

        // define boundary conditions of the spline
        id = 1;
        // psid = r = X[3]
        Zi0->e(q,id) = X->e(3); // fixed by IC

        ic++;
        ZiT->e(q,id) = c.e(ic); // end point derivative is free

//      printf("\nic = %d",ic);

        // spline interpolation //////////////////////
        interpolate(); // RHCOOL does this

        // calculate Xi,Ui based on Zi ///////////////
        calculate_XU(); // you programed this in RHC_user.cpp

        // integrate cost function ///////////////////

        // initial and final time
        t0 = t;
        tf = t+T;

        // calculate the cost function
        // J = 1/2 * int (x1^2 + x2^2 + u^2)
        // using trapezoidal integration
        // I = dt/2 * (f(t0) + 2*f(t1) + ... + 2*f(tn-1) + f(tn))
        dt = (tf-t0)/(Ni-1);
        sum = 0.0;
        for(i=2;i<Ni;i++) {
                x1 = Xi->e(1,i);
                x2 = Xi->e(2,i);
                x3 = Xi->e(3,i);
                x4 = Xi->e(4,i);
                x5 = Xi->e(5,i);
                x6 = Xi->e(6,i);
                u1 = Ui->e(1,i);
                u2 = Ui->e(2,i);

                tr = tb + ti->e(i);
                xr = Ax*cos(tr);
                yr = Ay*sin(w*tr);
//              xr = tr;
//              yr = xr*xr;

                L = 0*x1*x1 + 0*x2*x2 + 0*x3*x3 + 0*x4*x4 + (x5-xr)*(x5-xr)
                        + (x6-yr)*(x6-yr) + 0.0*u1*u1 + 0.0*u2*u2;

                sum += 2.0*L;
        }

        // add the endpoints

        i = 1;
        x1 = Xi->e(1,i);
        x2 = Xi->e(2,i);
        x3 = Xi->e(3,i);
        x4 = Xi->e(4,i);
        x5 = Xi->e(5,i);
        x6 = Xi->e(6,i);
        u1 = Ui->e(1,i);
        u2 = Ui->e(2,i);

        tr = tb + ti->e(i);
        xr = Ax*cos(tr);
        yr = Ay*sin(w*tr);
//      xr = tr;
//      yr = xr*xr;

        L = 0*x1*x1 + 0*x2*x2 + 0*x3*x3 + 0*x4*x4 + (x5-xr)*(x5-xr)
                + (x6-yr)*(x6-yr) + 0.0*u1*u1 + 0.0*u2*u2;
        sum += L;

        i = Ni;
        x1 = Xi->e(1,i);
        x2 = Xi->e(2,i);
        x3 = Xi->e(3,i);
        x4 = Xi->e(4,i);
        x5 = Xi->e(5,i);
        x6 = Xi->e(6,i);
        u1 = Ui->e(1,i);
        u2 = Ui->e(2,i);

        tr = tb + ti->e(i);
        xr = Ax*cos(tr);
        yr = Ay*sin(w*tr);
//      xr = tr;
//      yr = xr*xr;

        L = 0*x1*x1 + 0*x2*x2 + 0*x3*x3 + 0*x4*x4 + (x5-xr)*(x5-xr)
                + (x6-yr)*(x6-yr) + 0.0*u1*u1 + 0.0*u2*u2;
        sum += L;

        // calculate inetgral
        sum = dt/2.0*sum;

        // calculate cost function
        J = 0.5*sum;

//      printf("\nJ(debug) = %lf",J);

        return J;
}


RHC::compute_flat_ICs(dVector &x0, dVector &u0, dVector &ud0)
{

}


void f_heli_3dof2(double *X, double *U, double *Xd)
// heli model in local coord with combined expressions
// X[1] - u (velocity along body x axis)
// X[2] - v (velocity along body y axis)
// X[3] - r (angular velocity along body z axis)
// X[4] - psi (yaw euler angle - z axis)
// X[5] - xc (center of mass position in global coordinates)
// X[6] - yc (center of mass position in global coordinates)
{
        double u,v,ud,vd;
        double r,rd;
        double psi,psid;
        double xc,yc,xcd,ycd;

        double c_psi; // cos of angles
        double s_psi; // sin of angles

        // inputs
        double u_mr; // main rotor
        double u_tr; // tail rotor

        // parameters
        double m,Izz;
        double l_tr;
        double k_mr_x; // gain from main rotor thrust to x thrust component
        double S_fus_x,S_fus_y;
        double rho_a; // air density

        double Tmr; // main rotor thrust
        double Ttr; // tail rotor thrust

        double V_inf;
        double cx,cy;

        // set the helecoper paramters with the values from the report
        S_fus_x = 0.1;
        S_fus_y = 0.22;
        m = 8.2;
        Izz = 0.28;
        l_tr = 0.91;
        k_mr_x = -0.3;
        rho_a = 1.0;
        cx = -0.5*rho_a*S_fus_x;
//      cy = -0.5*rho_a*S_fus_y;
        cy = 0.0;

        // get state variables
        u = X[1];
        v = X[2];
        r = X[3];
        psi = X[4];
        xc = X[5];
        yc = X[6];

        // get inputs
        // U = [u_mr, u_tr]
        u_mr = U[1];
        u_tr = U[2];

        // assume the main rotor and tail rotor thrust are commanded by the inputs
        Tmr = u_mr;
        Ttr = u_tr;

        V_inf = sqrt(u*u + v*v);

        // flat outputs (z1,z2) = (u,psi)
        ud = v*r - k_mr_x * Tmr / m + cx * u * V_inf / m;

        // for this equation we can integrate it using Euler or RK
        // however ODE integration is time consuming and not
        // accutate compared to integration, so we assume cy = 0
        vd = -u*r + cy * v * V_inf / m;
        rd = -Ttr*l_tr/Izz;

        // euler angle derivative
        psid = r;

        // calculate some trig functions
        c_psi = cos(psi); s_psi = sin(psi);

        // global position derivatives
        // express velocity vl=[u,v] in global coord using Vg = R*vl
        xcd = c_psi*u - s_psi*v;
        ycd = s_psi*u + c_psi*v;

        Xd[1] = ud;
        Xd[2] = vd;
        Xd[3] = rd;
        Xd[4] = psid;
        Xd[5] = xcd;
        Xd[6] = ycd;
}
```

5.11 APPENDIX E: dVector.cpp

```
#include <cmath>   // math functions
#include <cstdio>  // standard I/O functions
#include <cstring> // string manipulation functions

// for real-time program remove some headers
#ifndef RTSS

#include <iostream>  // console stream I/O
#include <fstream>   // file stream I/O
#include <strstream> // string stream I/0

#include <conio.h> // console I/O functions such as getch()

using namespace std;

#endif

#include "dVector.h"

dVector::dVector(int n1, int n2)
{
        int i;

        safe_reference = 0.0;

        // initialize the member variables n1 and n2
        N1 = n1;
        N2 = n2;

        // initialize dynamic array pointer
        allocate_vector(A,N1,N2);

        if(A==NULL) {
                #ifdef DEBUG
                cout << "\nallocate_vector error in vector::vector(int n1, int n2)";
                #endif

                return;
        }

        // initialize array elements
        for(i=N1;i<=N2;i++) A[i] = (double)0.0;
}

dVector::~dVector()
{
        free_vector(A,N1,N2);
}


int allocate_vector(double *&a, int n1, int n2)
{
        // check for invalid arguments
        if(n2 < n1) {
                #ifdef DEBUG
                cout << "\nerror in allocate_vector: n2 < n1";
                #endif
                a = NULL;
                return 1;
        }

        // allocate a dynamic 1D array of doubles
        a = new double [ n2 - n1 + 1 ];

        if(a == NULL) {
                #ifdef DEBUG
                cout << "\nmemory allocation error in allocate_vector";
                #endif
                return 1;
        }

        a = a - n1;

        return 0;
}


int free_vector(double *&a, int n1, int n2)
{
        if(a == NULL) {
                #ifdef DEBUG
                cout << "\nNULL pointer in free_vector";
                #endif
                return 1;
        }

        a = a + n1;

        delete a;

        a = NULL;

        return 0;
}
```

5.12 APPENDIX F: dMatrix.cpp

```
#include <cmath>   // math functions
#include <cstdio>  // standard I/O functions
#include <cstring> // string manipulation functions

// for real-time program remove some headers
#ifndef RTSS

#include <iostream>  // console stream I/O
#include <fstream>   // file stream I/O
#include <strstream> // string stream I/O

#include <conio.h> // console I/O functions such as getch()

using namespace std;

#endif

#include "recipes.h"

#include "dVector.h"
#include "dMatrix.h"

dMatrix::dMatrix(int n1, int n2, int m1, int m2)
{
        int i,j;

        safe_reference = 0.0;

        N1 = n1; N2 = n2; M1 = m1; M2 = m2;

        allocate_matrix(A,N1,N2,M1,M2);

        if(A==NULL) {
                #ifdef DEBUG
                cout << "\nallocate_matrix error in dMatrix::dMatrix(int n1, int n2, int
m1, int m2)";
                #endif

                return;
        }

        // initialize the array elements
        for(i=N1;i<=N2;i++) {
                        for(j=M1;j<=M2;j++) {
                                A[i][j] = (double)0.0;
                        }
        }
}


dMatrix::~dMatrix()
{
        free_matrix(A,N1,N2,M1,M2);
}


int allocate_matrix(double **&a, int n1, int n2, int m1, int m2)
// allocate a dynamic 2D array
{
        int i,j;

        // check for invalid arguments
        if(n2 < n1) {
                #ifdef DEBUG
                cout << "\nerror in allocate_dMatrix: n2 < n1";
                #endif
                a = NULL;
                return 1;
        }

        // check for invalid arguments
        if(m2 < m1) {
                #ifdef DEBUG
                cout << "\nerror in allocate_dMatrix: m2 < m1";
                #endif
                a = NULL;
                return 1;
        }

        // allocate a dynamic array of pointers
        a = new double * [n2-n1+1];

        // check for memory allocation error
        if(a==NULL) {
                #ifdef DEBUG
                cout << "\nmemory allocation error #1 in allocate_matrix";
                #endif
                return 1;
        }

        a = a - n1; // make the array go from n1 to n2

        // initialize each pointer in the array a to a 1D array that stores each row
        for(i=n1;i<=n2;i++) {

                a[i] = new double [m2-m1+1]; // declare a 1D array for each row

                // if there is an allocation error we need to free memory and return NULL
                if(a[i] == NULL) {
                        for(j=n1;j<i;j++) delete (a[j]+m1); // delete each row up to i-1

                        delete (a+n1); // delete the array of pointers

                        #ifdef DEBUG
                        cout << "\nmemory allocation error #2 in allocate_dMatrix";
                        #endif

                        a = NULL;
                        return 1;
                }

                a[i] = a[i] - m1; // adjust for the column offset m1
        }

        return 0;
}


int free_matrix(double **&a, int n1, int n2, int m1, int m2)
// free a dynamic 2D array of doubles
{
        int i;

        // check if a is a NULL pointer
        if(a==NULL) {
                #ifdef DEBUG
                cout << "\nerror: NULL pointer in free_matrix";
                #endif
                return 1; // error
        }

        for(i=n1;i<=n2;i++) delete (a[i]+m1); // delete each row

        delete (a+n1); // delete the array of pointers

        a = NULL; // set the pointer equal to NULL

        return 0; // OK
```

```
}


void add(dMatrix &A, dMatrix &B, dMatrix &C)
// C = A + B
{
        int i,j;

        for(i=A.N1; i<=A.N2; i++) {
                for(j=A.M1; j<=A.M2; j++) {
                        C.e(i,j) = A.e(i,j) + B.e(i,j);
                }
        }
}


void add(dVector &a, dVector &b, dVector &c)
// c = a + b
{
        int i;

        for(i=a.N1; i<=a.N2; i++) c.e(i) = a.e(i) + b.e(i);
}


void sub(dMatrix &A, dMatrix &B, dMatrix &C)
// C = A - B
{
        int i,j;

        for(i=A.N1; i<=A.N2; i++) {
                for(j=A.M1; j<=A.M2; j++) {
                        C.e(i,j) = A.e(i,j) - B.e(i,j);
                }
        }
}


void sub(dVector &a, dVector &b, dVector &c)
// c = a - b
{
        int i;

        for(i=a.N1; i<=a.N2; i++) c.e(i) = a.e(i) - b.e(i);
}


void mult(dMatrix &A, dMatrix &B, dMatrix &C)
// multiply two matrices
// C = A * B
{
        int i,j,k;

        for(i=A.N1; i<=A.N2; i++) {
                for(j=B.M1; j<=B.M2; j++) {
                        C.e(i,j) = (double)0.0;
                        for(k=A.M1; k<=A.M2; k++) {
                                C.e(i,j) += A.e(i,k) * B.e(k,j);
                        }
                }
        }
}


void mult(dMatrix &A, dVector &b, dVector &c)
// multiply a matrix times a vector
// c = A * b
{
        int i,j;

        for(i=A.N1; i<=A.N2; i++) {
                c.e(i) = (double)0.0;
                for(j=A.M1; j<=A.M2; j++) {
                        c.e(i) += A.e(i,j) * b.e(j);
                }
        }
}


void cross(dVector &a, dVector &b, dVector &c)
// c = a x b
// all vectors are 1x3
{
        c.x() = a.y() * b.z() - b.y() * a.z();
        c.y() = b.x() * a.z() - a.x() * b.z();
        c.z() = a.x() * b.y() - b.x() * a.y();
}


void matrix_cross(dVector &v, dMatrix &V)
// matrix representation of cross product
// v - input vector (1x3)
// V - matrix representation of cross product (3x3)
{
        V.e(1,1) = (double)0.0;
        V.e(1,2) = -v.z();
        V.e(1,3) = v.y();

        V.e(2,1) = v.z();
        V.e(2,2) = (double)0.0;
        V.e(2,3) = -v.x();

        V.e(3,1) = -v.y();
        V.e(3,2) = v.x();
        V.e(3,3) = (double)0.0;
}


void transpose(dMatrix &A, dMatrix &B)
// B = A'
{
        int i,j;

        for(i=A.N1; i<=A.N2; i++) {
                for(j=A.M1; j<=A.M2; j++) {
                        B.e(i,j) = A.e(j,i);
                }
        }
}

void lu_dcmp(dMatrix &A, dMatrix &A_lud, int *indx)
// Calculate the LU decomposition of a nxn matrix a
// A_lud[1..n][1..n] - LU decomposition of matrix A
// indx[n+1] = - row permutation index
// It assumes 1 offset arrays.
{
        int i,j;
        double *d,x;

        d = &x;

        // A_lud = A
        for(i=1;i<=A_lud.N2;i++) {
                for(j=1;j<=A_lud.M2;j++) {
                        A_lud.A[i][j] = A.A[i][j];
                }
        }

        ludcmp(A_lud.A, A.N2, indx, d);
}
```

```
void lu_solve(dMatrix &A_lud, int *indx, dVector &b, dVector &x)
// Return the solution vector x of Ax=b using the
// LU decomposition of a matrix.
// It assumes 1 offset arrays.
{
        int i;

        // x = b
        for(i=x.N1;i<=x.N2;i++) x.A[i] = b.A[i];

        lubksb(A_lud.A, A_lud.N2, indx, x.A);
}


void sv_dcmp(dMatrix &a, dMatrix &u, dVector &w, dMatrix &v)
// Calculate the Singular Value Decomposition
// of a matrix A = UWV'.
// a - the mxn matrix A              u - the mxn matrix U
// v - the nxn matrix V              w - the n vector W
{
        int i,j;

        // u = a
        for(i=1;i<=a.N2;i++)
                for(j=1;j<=a.M2;j++) u.e(i,j) = a.e(i,j);

        svdcmp(u.A, a.N2, a.M2, w.A, v.A);
}


void sv_solve(dMatrix &a, dVector &b, dMatrix &u, dVector &w, dMatrix &v, dVector &x)
// Return the solution vector of Ax=b using the SV
// decomposition of a matrix.
// a - the mxn matrix A              u - the mxn matrix U
// v - the nxn matrix V              w - the n vector W
// x - the n solution vector
{
        int i;
        double tol=1.0e-5, wmax=0.0, thresh;

        // threshold singular values
        for(i=1;i<=a.M2;i++) if(w.e(i)>wmax) wmax = w.e(i);
        thresh = tol*wmax;
        for(i=1;i<=a.M2;i++) if(w.e(i)<thresh) w.e(i)=0.0;

        svbksb(u.A, w.A, v.A, a.N2, a.M2, b.A, x.A);
}

double cond(dMatrix &a)
// Find the condition number of a matrix using sv_dcmp
{
        int i,j;
        double wmax, wmin;

        dVector w(1,a.M2);
        dMatrix u(1,a.N2,1,a.M2), v(1,a.M2,1,a.M2);

        // u = a
        for(i=1;i<=a.N2;i++)
                for(j=1;j<=a.M2;j++) u.e(i,j) = a.e(i,j);

        svdcmp(u.A, a.N2, a.M2, w.A, v.A);

        wmin = wmax = w.e(1);
        for(i=2;i<=a.M2;i++) if(w.e(i)>wmax) wmax = w.e(i);
        for(i=2;i<=a.M2;i++) if(w.e(i)<wmin) wmin = w.e(i);

        return (wmax/wmin);
}
```

5.13 G: dMatrix_array.cpp

```
#include <cmath>    // math functions
#include <cstdio>   // standard I/O functions
#include <cstring> // string manipulation functions

// for real-time program remove some headers
#ifndef RTSS

#include <iostream>  // console stream I/O
#include <fstream>   // file stream I/O
#include <strstream> // string stream I/0

#include <conio.h> // console I/O functions such as getch()

using namespace std;

#endif

#include "recipes.h"

#include "dVector.h"
#include "dMatrix.h"

#include "dMatrix_array.h"

dMatrix safe_dmatrix0(1,3,1,3);

dMatrix_array::dMatrix_array(int k1, int k2, int n1, int n2, int m1, int m2)
{
        int k;

        safe_dMatrix = &safe_dmatrix0;
        safe_reference = 0.0;

        K1 = k1; K2 = k2;
        N1 = n1; N2 = n2;
        M1 = m1; M2 = m2;

        M = new dMatrix * [ k2 - k1 + 1 ];
        if(M == NULL) {
                #ifdef DEBUG
                cout << "\nmemory allocation error in dMatrix_array";
                #endif
                return;
        }
        M = M - k1;

        for(k=K1;k<=K2;k++) M[k] = NULL;

        for(k=K1;k<=K2;k++) {
                M[k] = new dMatrix(n1,n2,m1,m2);
                if(M[k] == NULL) {
                        #ifdef DEBUG
                        cout << "\nmemory allocation error in dMatrix_array";
                        #endif
                        return;
                }
        }
}


dMatrix_array::~dMatrix_array()
{
        int k;

        if (M == NULL) return;
        for(k=K1;k<=K2;k++) {
                if( M[k] != NULL ) delete M[k];
        }

        M = M + K1;
        delete M;
}
```

5.14 Appendix H: Explanation of Attachments

In this chapter, the experimental results and the codes written for the materials of this report which are included in the attached CD is described. The data is classified based on the different chapters of this report.

Chapter 3

All of the data used for chapter 3 of the report are included in this folder. It includes different subfolders based on the report and the contents of each folder are described bellow.

Section 3.2

All of the supporting data for section 3.2 of the report is included in this folder. It has the following subfolders:

- **3.2.1**: The data used for the Figures 10 to 17 are collected in this folder with name "3.2.1". This folder is consists of three parts with the following names:
  - *Part one*: The data used in section 3.2.1.1 of the report is in this folder. It consists of four folders named "data1", "data4", "data7", and "data9". Each of them has a C++ code for processing data to identify the model parameters of Hovercraft in the Forward Movement. By running this code the values b_1', a_1 and a_2 of equation (28) are calculated using Singular Value Decomposition (SVD). For plotting the results "plot_fig.m" should be run using MATLAB 6.5[31].
  - *Part Two*: The data used in section 3.2.1.2 of the report is in this folder. It consists of one folder named "data2" which has a C++ code for processing data to identify the model parameters of Hovercraft in the Rotation Direction. By running this code the values b_3', a_3 and a_4 of equation (30) is calculated using SVD. For plotting the results "plot_fig.m" should be run.
  - *Part Three*: The data used in section 3.2.1.3 of the report is in this folder. It contains two folders named "Forward Movement" and "Side Movement".
    - *Forward Movement* has two folders "data5" and "data8". Each of them has a C++ code for processing data to identify b_1' of equation (28) using SVD. For plotting the results "plot_fig.m" should be run.
    - *Side Movement* has one folder named "data2" which contains a C++ code for processing data to identify b_2' of equation (29) using SVD. For plotting the results "plot_fig.m" should be run.

- **3.2.2**: In this folder, all of the results used in the section 3.2.2 of the report are presented. It has the same folders as previous folder (3.2.1) but all of the data used

[31] All of the *.m files in this report should be run by MATLAB 6.5.

to find the model parameters are included, so the main folders are the same as folder 3.2.1 but these folders include more data.

Section 3.3

For this folder "RHCOOL Version 2" is included. It solves the example of 3DOF helicopter and simulates it. The path followed by the 3DOF helicopter in this example is an ellipse.

Chapter 4

This folder includes some subfolders which are described bellow and each of the subfolders has one "readme.doc" file which describes how the data is stored in all of the data files of that subfolder. If somebody wants to do more process on the data file of that subfolder, he/she should refer to that readme.doc file.

Section 4.1

This folder includes one readme file described before and one folder named "4.1.1.2". The experimental data of Wireless InertiaCube3 explained in section 4.1.1.2 of the report, is included in that folder. It has all data files and by running "plot_all_data.m" Figure 28, Figure 30, Figure 32, and Figure 33 of the report will be regenerated.

Section 4.2

This folder includes one readme file described before and two folders named "4.2.1" and "4.2.2" as follows:

- **4.2.1**: By running "plot_f_m.m" the Figure 39 which is related to the Forward movement of hovercraft is produced. The original data were measured by PCIBird [23].
- **4.2.2**: By running "plot_rotation.m" the Figure 40 which is related to the Rotational movement of hovercraft is produced. The data in this part, measured by wireless InertiaCube3.

Section 4.3

This folder includes one readme file described before and two folders named "4.3.1" and "4.3.2" as follows:

- **4.3.1**: this folder includes two folders:
    - *Regulation with RHC*: the original data of Figure 42 of the report is in this folder. In addition by running "plot_data.m" Figure 42 can be regenerated.
    - *Tracking with RHC*: the original data of Figure 43 of the report and a Clip showing the hovercraft while tracking a sinusoidal path are in this folder. In addition by running "plot_data.m" Figure 43 can be regenerated.

- **4.3.2**: this folder includes two folders as bellow:

- *Vision_Inertiacube3*: this folder includes the original data of Figure 44. This results show the tracking of one hovercraft while the orientation of hovercraft (ψ) is measured by vision system and the angular velocity ($\dot{\psi}$) is measures by wireless InertiaCube3. Figure 44 can be regenerated by running "plot_data.m".
- *only_vision*: the original data of Figures 45 and 46 are in this folder. In this case all of the data required for controller are measured by vision system. These two figures can be regenerated by running "plot_data.m".

Section 4.4

This folder includes three folders as bellow:

- **Scheduling_two_systems**: this folder includes one readme file which describes how the data is stored in "data.txt". By running "plot_data.m" the Figure 48 can be regenerated. This folder also includes another folder named "scheduling_video_clips" which contains three folders named "Video1", "Video2", and "Video3". In each of them one video clip of scheduling two hovercrafts in one computer, with the data file and "plot_data.m" are exist.
- **System_1**: this folder includes one readme file which describes how the data is stored in "data.txt". By running "plot_data.m" the Figure 49 can be regenerated.
- **System_2**: this folder includes one readme file which describes how the data is stored in "data.txt". By running "plot_data.m" the Figure 50 can be regenerated.

Chapter 6

In this folder, some of the references are included for more convenience of the reader.

CHAPTER 6: A VARIABLE COMMUNICATION APPROACH FOR DECENTRALIZED RECEDING HORIZON CONTROL OF MULTI-VEHICLE SYSTEMS

(Task #10)

Hojjat A. Izadi
PhD Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory
Concordia University, Montreal, Quebec, H3G 1M8, Canada

Jan 2007

Abstract

This report documents the work related to task #10 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". Task#10 involves theoretical verification of the feasibility and stability of the decentralized receding horizon control (RHC) for multi vehicle systems. The stability and performance of decentralized RHC is often improved by modifying the cost function and constraints. However, variable communication presents an additional approach for further improvement. In this report, using variable communication a new decentralized RHC algorithm is developed. In spite of the other decentralized RHC methods that try to reduce the communication requirements, the proposed method allows utilizing the maximum available communication capacity for further improvement of the stability and performance. Through modification of stability conditions developed for time varying decentralized RHC, an investigation of the effect of 1- faster communication rates, 2- expanded communication radius, and 3- multi-hopping communication on decentralized RHC is performed. Then, the new results are used to find bounds on RHC parameters such as prediction horizon and execution horizon. This provides a more systematic way for choosing decentralized RHC parameters so that stability is guaranteed. Also, since the communication induces a delay to the system, a new formulation of decentralized RHC is proposed which accounts for large communication delays. Two different case studies on the formation of mobile robots and formation flight of UAVs have been investigated to clarify the problem statement.

Nomenclature

A, B :	Linear system matrix
E :	Set of edges (neighbour vehicles)
G :	Graph topology
H :	Hamiltonian equation
I_{zz} :	Yawing Moment of inertia
J :	Cost function
L_x, L_u :	Lipchitz constant
l_{tr} :	Tail rotor hub location behind c.g
N_v :	Number of vehicles
$P, Q, and\ R$:	Weight matrices of quadratic cost function
q :	Costate Vector
R_c :	Communication radius
R_s :	Safety radius
r: :	Yawing Rate
S_x^{fus} :	Frontal fuselage drag area
T :	Prediction horizon (or finite horizon time)
u :	vector of control inputs
V :	Set of vehicles
x :	Vector of states
β :	Model uncertainty
ψ :	Euler angle (Yaw angle)
δ :	Execution horizon (sampling time)
μ :	Uncertainty coefficient
τ_d :	Communication delay

Superscript

* = Optimal value of parameter

Subscript

actual = Actual system
predicted = Predicted value of parameter

6.1 Introduction

This report addresses the following task related to the project "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques" and the following task:

Task 10: To validate theoretically the feasibility and existence of solutions, the stability, and the robustness of the RHC despite an adversarial and partially unknown environment.

Over the past few decades there has been significant interest in decentralized control of cooperative problems[1-6]. The decentralized structure potentially results in increased autonomy of agents while providing some sort of cooperation among the agents to perform a common mission. In fact, the main idea with decentralized control is to break the centralized controller into the local control problems of smaller sizes in order to reduce the computation effort and communication requirements.

However, decentralization must provide and respect the primary objectives of cooperative control problem, these primary objectives are: 1-The total cost added due to cooperation must provide a greater increase in expected system effectiveness than the case of the non-cooperating vehicles. 2- Performance lower bound of a cooperative system as communication degrades should never be worse than the performance of the same vehicles without cooperation. Since these primary goals of the cooperative architectures are not achieved yet through decentralization, the researchers are motivated to investigate new potential ways for further improvements of the decentralized control architectures. For example, most recently with recent advances in distributed computation, there have been numerous attempts to use computational expensive optimization based control methods such as receding horizon control (RHC)[7,8] for decentralizing the cooperative problems. Although RHC is computationally expensive, it has some prominent capabilities which motivate the researchers to develop the RHC based decentralized control architectures[1, 2, 6, 7]. For example, RHC readily handles systems constraints and saturations while providing optimal control action. Also, since in RHC the control action is generated through minimization of a cost function it is easy to provide a coupling between neighbour vehicles in the cost function or constraints.

Another potential way, which is also one step ahead, is to use the communication based approaches for stability enhancement of the decentralized RHC architectures; this approach is the main focus of this report. Using the variable communication, it is desired to use the maximum available communication capacity to enhance the stability and performance of the decentralized RHC architecture. Although most of the research works in the field of decentralized RHC have focused on modifying the cost function and constraints, they lack utilizing communication methods efficiently for further enhancement of stability. It is also noticeable that since most of these research works persist on reducing the communication requirement they sacrifice the benefits of communication capabilities and don't allow using advanced communication technologies. Hence, new decentralized RHC algorithms are required to utilize the maximum available communication capacity. Since in the literatures this mechanism is not thoroughly addressed, in this report, the role of the communication for improving the stability of the decentralized RHC algorithm is investigated. Also, an algorithm is proposed which enables vehicles to find the best combination of communication topology and RHC parameters while the stability is ensured.

In the decentralized RHC, a centralized RHC controller is broken into the local control problems of smaller sizes. To capture the nature of cooperation among vehicles, a coupling between neighbour vehicles is provided in the formulation either in the cost function[1, 2] or in the constraints[6]. However, control actions computed locally are not guaranteed to be globally feasible in general[1] and to stabilize the team. In general, stability and feasibility of the decentralized RHC methods are difficult to prove and may yield poor performance[10-12].

As in classical RHC, most of the researchers have addressed the stability and feasibility of the decentralized RHC by modifying the cost function and constraints. For example, a sophisticated work is conducted by Keviczky et al.[1], where a decentralized RHC is proposed for decoupled discrete time systems by breaking down a centralized RHC architecture into the distinct RHC controllers of smaller sizes. Each RHC controller is associated to a different vehicle and computes the local control inputs based only on the states of itself and its neighbours. In this approach, the vehicles are coupled through a cost function and the necessary information from the neighbour vehicles is provided through non-delay communication or measurement. Each vehicle predicts its neighbours' behaviour from their dynamical model and based on this prediction plans the trajectory of itself and its neighbours, but executes its own trajectory and discards the trajectories of neighbours. It is proven that if the mismatch between the predicted and actual trajectories of all neighbours is smaller than some cost function related to the initial conditions then the global group stability is achieved. In another work, Dunbar et al. proposed a distributed RHC for multi vehicles with continuous time dynamically decoupled subsystems whose state vectors are coupled through the cost function of a RHC control problem[2]. Each vehicle solves an optimization problem and generates its control action using an assumed control action of neighbour vehicles. The key requirement for stability is that each distributed optimal control not deviate too far from the previous one; this constraint restricts the optimization problem and doesn't allow very fast maneuvers. Also, the communication delay is not addressed.

As an example of constraint coupled decentralized RHC architecture one can refer to [6] and [10] where authors decentralized a previously developed robust safe but knowledgeable (RSBK) model predictive control algorithm [11] that uses the constraint tightening technique to achieve robustness and lower online computation burden. Using local knowledge of the group, it is ensured that each vehicle always has a solution guaranteeing robust feasibility of the entire fleet under the action of disturbances. The algorithm is extended in [12], to explicitly account for the communication and computation delays in the system in the presence of bounded disturbances.

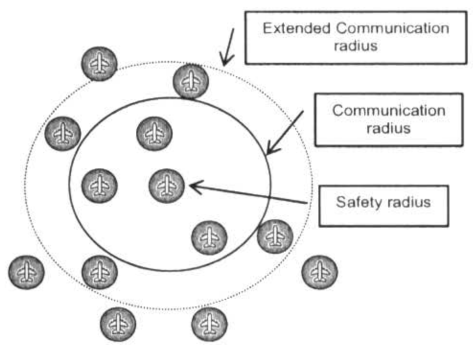


Fig. 1: Communication radius and safety radius

Our analysis shows that combining the stability results with variable communication can lead to stability enhancement of the decentralized RHC approach for more realistic cases. To discuss the variable communication methods, some concepts regarding the communication such as communication radius and multi-hopping communication are introduced and investigated to exploit this mechanism. The communication radius (figure 1) defines a region around each vehicle where communication with vehicles which are inside this region is possible. Multi-hopping communication will be introduced later in section 5.1. In this report, it is studied how the following three techniques can enhance the stability of the decentralized RHC method: 1- faster communication rates 2- enlarging the communication radius, 3- multi-hopping communication. Based on these investigations a communication based algorithm is proposed for further stability enhancement of decentralized RHC. The main idea with the proposed method is to improve the communication topology and select the decentralized RHC parameters through using the maximum available communication capacity so that the overall stability and performance is guaranteed.

The main contributions of this report are presented as follows: first a new formulation of decentralized RHC is established for continuous-time dynamically decoupled systems. This new formulation accounts for communication delays larger than sampling time. Also, this formulation allows using a time varying graph topology along with a time varying cost function for representing a time varying coupling, cooperation and communication topology among vehicles. Secondly, a stability condition is derived for the new time varying decentralized RHC formulation; through driving the stability condition a bound is found on the differentiation of the cost function which is treated as the Lyapanov function candidate. This bound allows defining a stability margin to have a measure of stability and performance. Also, this stability condition allows studying the variable communication. Then, based on the derived stability condition, the effect of faster communication rate, larger communication radius, and multi-hopping communication on the stability is investigated. Using the extension of the Bellman-Grownwall lemma along with the advantages of Hamiltonian equation, the acceptable region for the *prediction horizon* and *execution horizon* (communication rate) are determined that allows tuning RHC in a systematic way so that the stability is guaranteed. The results of these analyses finally lead to proposing a new decentralized RHC algorithm which allows choosing the optimum communication topology along with the sampling time and prediction horizon. This algorithm enables each vehicle to use the maximum available communication capacity for stability enhancement of decentralized RHC architecture. The last part involves two examples which clarify the connection between the proposed approach and realistic scenarios. The first example is the simulation of a group of 5 robots that verifies the effectiveness of proposed method for enhancement of the stability and performance by modifying the communication topology. And, the second example involves the simulation of a fleet of three rotorcraft UAVs; the selected scenario shows how in the real cases we need the variable communication to perform the missions. More precisely, in the selected scenarios the vehicles needs to change the communication topology in order to avoid obstacles or visiting the far targets. The contribution of this research work has been four conference papers so far [13-16].

6.2 Problem Formulation

6.2.1 Problem Statement

As it is mentioned previously, the main objective of this report is to develop a new algorithm so that the maximum available communication capacity can be used for stability enhancement of decentralized RHC architecture. More specifically, an algorithm is developed which optimizes the communication topology, communication rate and prediction horizon at each time step using variable communication. The reason for using the variable communication is to cover a more general set of real scenarios; because in the real scenarios a fixed communication topology can not be used most of the time. For example, figure 2 shows a formation problem in which the UAVs leave the formation due to visiting the far targets. When the vehicles flight in the formation they can communicate with each other but when they leave the formation they may get beyond the communication radius and won't be able to communicate with each other. Hence, a variable communication topology is needed in this case for mathematical representation and studying of scenario.

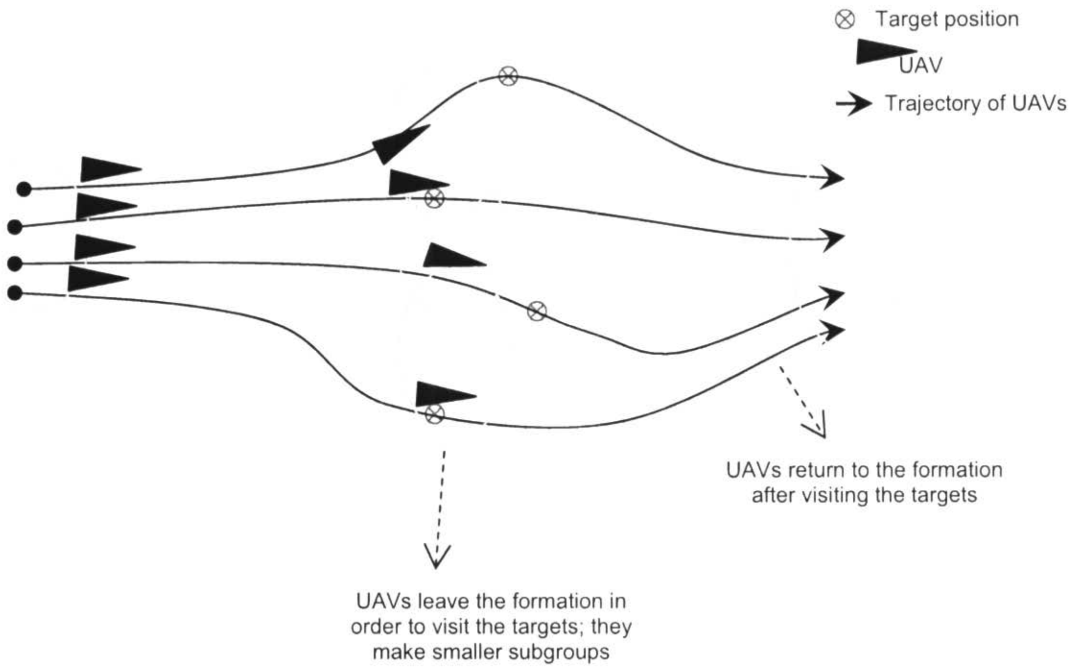

Target position
UAV
Trajectory of UAVs
UAVs return to the formation after visiting the targets
UAVs leave the formation in order to visit the targets; they make smaller subgroups

Fig. 2: Formation of a fleet of UAVs: a time varying communication topology due to far targets

The same scenario may happen where the obstacles and non-fly zones in the battle field drive the vehicles to go far away of each other and change the graph topology. In this report one example of such problems is discussed.

The following diagram shows how this report is organized to develop the new algorithm and how the result of each section is related to other sections:

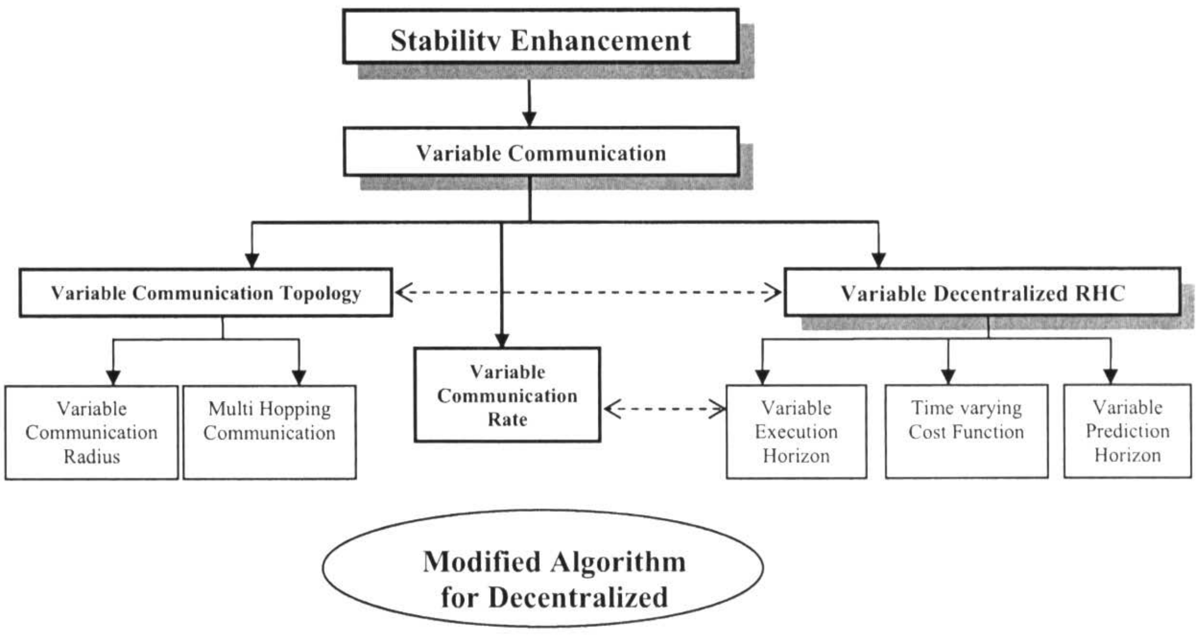

Stabilitv Enhancement
Variable Communication
Variable Communication Topology
Variable Decentralized RHC
Variable Communication Radius
Multi Hopping Communication
Variable Communication Rate
Variable Execution Horizon
Time varying Cost Function
Variable Prediction Horizon
Modified Algorithm for Decentralized

Fig. 3: Schematic representation of the proposed approach for stability enhancement

This diagram says that to enhance the stability a variable communication method is used. A variable communication requires that the communication topology, communication rate and decentralized RHC be time varying as well. The following explains how these variations interact and affect each other:

1- Variable Communication Topology: Variable communication topology is used whenever the stability and performance of the decentralized RHC are not satisfactory. Although the measurement of stability and performance is not the subject of this report, a stability margin is defined in this report to have a measure of stability. The measure of performance depends on the specification of each problem, requested mission and desired performance index. For example, in a path planning problem it is desired to visit some targets without colliding the obstacles; hence, the measure of performance comes from the fact that how close the vehicle get to the target and how better they can respect the limitations.

Once the stability and performance of the decentralized RHC are not satisfactory a modification in the communication topology is sought through: 1- Changing the communication radius and 2- Using the multi-hopping communication. This variation in the communication topology imposes that the decentralized RHC architecture be time varying because the decentralized RHC is formed at each update based on the current communication topology.

2- Variable Decentralized RHC: A time varying flexible formulation of decentralized RHC is required for representation of time-varying parameters. This time varying formulation is discussed in section 3.2 and includes the time varying: 1- Cost function, 2- *Prediction horizon* and 3- *Execution horizon.* Since the *execution horizon* is equal to the communication sampling time, the time variation of *execution time* implies the time variation of the communication sampling time; hence, the communication rate must be also taken into account. The related algorithm for online implementation of this time-varying formulation is provided in section 3.3 and is extended in 3.4 to account for communication delay.

3- Parameter Selection: Once a topology is selected it is required to answer two major questions: 1- What is the best communication rate? 2- What is the best prediction horizon for decentralized RHC? These two questions are answered in section 5.2 and 5.3 for the case of model uncertainty and no model uncertainty. For each communication topology there exist an optimum value of communication rate and prediction horizon, these values must respect the stability condition of decentralized RHC and communication limitations. Hence a stability analysis is required for the variable decentralized RHC architecture so that it yields a suitable stability condition. After modification of the communication topology the results of this section will be used to select the corresponding optimum value of the sampling time and prediction horizon.

4- Modified Algorithm for Decentralized RHC: Based on the results from the variable communication and stability analysis of the decentralized RHC a new algorithm is proposed for the online optimization of the communication topology so that the stability enhancement is achieved through using the maximum available communication capacity.

To address the mentioned issues we follow the steps below:

1- Formulation of time varying communication topology and cost function.

2- Formulation of time varying decentralized RHC considering the case of communication delay.

3- Stability analysis of time varying decentralized RHC and providing a stability condition and stability margin to have a measure of stability.

4- Investigating the effect of expanding the communication radius on the stability margin.

5- Investigating the effect of multi-hopping communication on the stability margin.

6- Investigating the effect of *prediction horizon* on the stability margin.

7- Investigating the effect of faster communication rate (execution horizon) on the stability margin.

8- Proposing an algorithm for optimum selection of communication topology, communication rate and prediction horizon.

9- Proposing a modified decentralized RHC algorithm.

These steps are shown in the following diagram and the results of each step are mentioned in the right side of each box, the dotted arrows shows the relations between the results of each step and how the results drive each other:

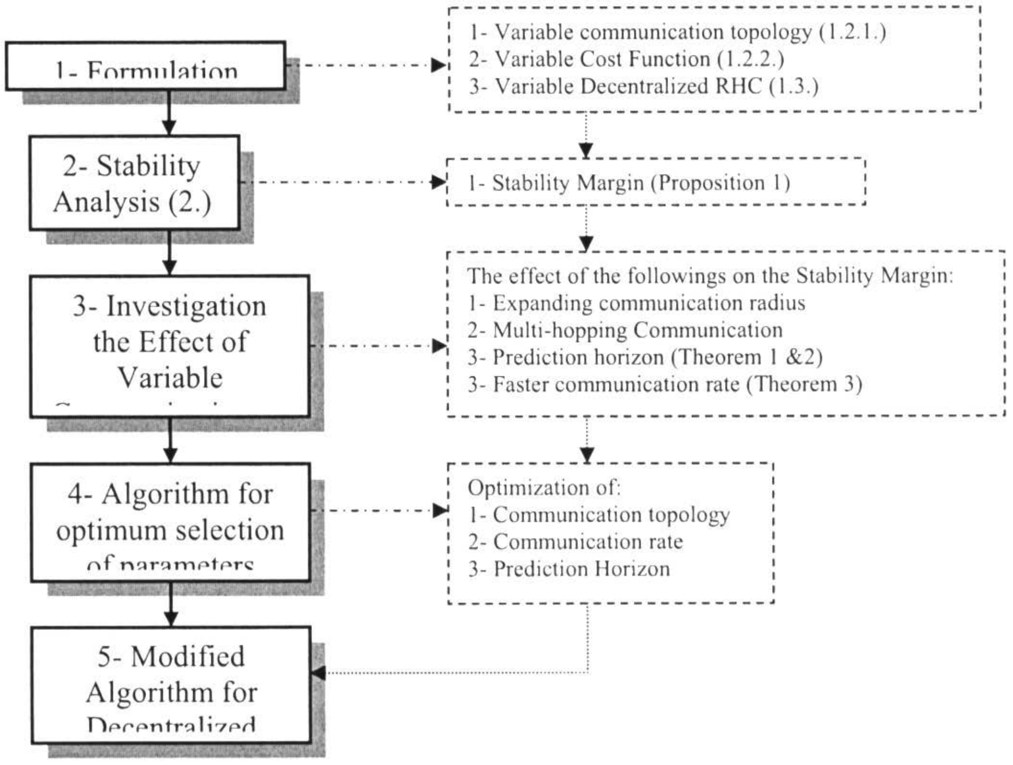

1- Formulation
2- Stability Analysis (2.)
3- Investigation the Effect of Variable
4- Algorithm for optimum selection of parameters
5- Modified Algorithm for Decentralized
1- Variable communication topology (1.2.1.)
2- Variable Cost Function (1.2.2.)
3- Variable Decentralized RHC (1.3.)
1- Stability Margin (Proposition 1)
The effect of the followings on the Stability Margin:
1- Expanding communication radius
2- Multi-hopping Communication
3- Prediction horizon (Theorem 1 &2)
3- Faster communication rate (Theorem 3)
Optimization of:
1- Communication topology
2- Communication rate
3- Prediction Horizon

Fig. 4: Schematic representation of the proposed approach for stability enhancement

6.2.2 Decentralized RHC Formulation

6.2.2.1 Time varying communication topology:

A time varying graph topology is introduced to present the coupling and cooperation among the vehicles. Consider a set on N_v vehicle cooperating to perform a common mission, the i-th vehicle is associated to the i-th node of the graph and if an edge (i, j) connecting the i-th and j-th node is present, it means the i-th and j-th vehicle are neighbour. This leads to an interconnection graph as follows:

$$G(t) = \{V, E(t)\} \tag{1}$$

Where, V is the set of nodes (vehicles) and $E(t) \subseteq V \times V$ the set of time varying edges (i, j), with $i, j \in V$. The interconnection graph is undirected i.e. $(i, j) \in E(t)$ implies $(j,i) \in E(t)$ even though it is not appear in $E(t)$. Also, the interconnection constraint between neighbour vehicles in the group (the i-th and the j-th vehicles) is presented as follows:

$$g_{i,j}(x_i, x_j) \leq 0 \tag{2}$$

Different ways for defining the neighbour set and connectedness of graph topologies have been investigated in [17]. Here we assume that the time dependence of the set of edges is a function of the relative distance of the vehicles; hence, the set *E(t)* is defined as follows:

$$E(t) = \{(i, j) \in V \times V \,|\, d_{ij} \leq R_C\} \tag{3}$$

where, d_{ij} denotes the distance between i-th and j-th vehicles, and R_C is the communication radius of vehicles (figure **1**). Furthermore, to ensure the collision avoidance it is required to:

$$d_{ij} \geq R_s \tag{4}$$

where, R_s is the safety radius around each vehicle and it is shown in figure **1**. The safety radius defines a region around each vehicle that neighbours aren't allowed to get inside this region otherwise the collision may occur. This constraint is added to the formulation of the decentralized RHC problem to prevent the vehicles from colliding each other. Defining the following set for each vehicle helps representing the coupling in the next section:

$$E^i(t) = \{(i, j) \in V \times V \,|\, d_{ij} \leq R_C^i(t)\} \tag{5}$$

Where $R_C^i(t)$ is the communication radius of i-th vehicle at time t. This time varying graph topology enables us to represent all time varying configurations of the subgroups.

6.2.2.2 Time varying cost function:

The vehicles may also be coupled in the cost function; then to cover a generic set of cooperative control problems, the following time varying cost function is proposed for the i-th vehicle:

$$J^i_{T^i(t)}(\tilde{x}^i(t),\tilde{u}^i(\cdot)) = L^i_{T^i(t)}(x^i(t),u^i(\cdot)) + \sum_{j|(i,j)\in E(t)} L^j_{T^i(t)}(x^j(t),u^j(\cdot)) \qquad \textbf{(6)}$$

Where, $T^i(t)$ is the *prediction horizon* of i-th vehicle at time t and:

$$\begin{aligned} \tilde{x}^i(t) &= \{x^i(t)\} \cup \{x^j(t) | (i,j) \in E(t)\} \\ \tilde{u}^i(t) &= \{u^i(t)\} \cup \{u^j(t) | (i,j) \in E(t)\} \end{aligned} \qquad \textbf{(7)}$$

The first term in (6) is associated to the individual vehicle and the second term is associated to the neighbour vehicles and represents the coupling between the i-th vehicle and its neighbours i.e. it is defined on the set of neighbours of i-th vehicle. Since $E(t)$ is time varying then the cost function becomes also time varying.. The cost function $L^i_{T^i(t)}$ is defined over the prediction horizon time $T^i(t)$ for each vehicle.

Remark: In a centralized RHC problem the cost function is formed as the summation of all individual cost functions ($L^i_{T^i(t)}$) of all vehicles. In fact, the optimization problem is solved for all vehicles by a central agent and the planned trajectory for each vehicle is transmitted to it.

In our case the cost function for each vehicle is defined as following for the dynamically decoupled continuous time systems:

$$L^i_{T^i(t)}(x^i(t),u^i(\cdot)) = \int_t^{t+T^i(t)} (\|x^i_t(\tau)\|^2_Q + \|u^i_t(\tau)\|^2_R) d\tau + \|x^i_t(T)\|^2_P \qquad \textbf{(8)}$$

Where, $x^i_t(\tau)$ and $u^i_t(\tau)$ are the state vector and control vector of i-th vehicle respectively at time τ, calculated from the optimization problem started at time t. Also, P, Q and R are positive definite and symmetric matrix penalties. This kind of cost function allows formulation of more generic set of problems and is used widely in the literatures [1, 2].

Moreover, this it is possible to benefit from the unique properties of quadratic cost functions in the optimization problems[18].

6.3 Variable Decentralized RHC:

In the RHC method a cost function is optimized over a finite time called *prediction horizon,* T, and the first portion of the computed optimal input is applied to the plant during a period of time called the *execution horizon,* δ; repeating this procedure yields a closed loop solution, the reader is referred to [9] for a comprehensive review of RHC literatures. Suppose that the following represents the nonlinear decoupled dynamics of the i-th vehicles:

$$\dot{x}^i(t) = f(x^i(t), u^i(t)) \tag{9}$$

We will define optimization problem $P^i(t)$ for the i-th vehicle as follows:

Variable Decentralized RHC Problem $P^i(t)$:

$$\min_{\tilde{u}^i(\cdot)} \; J^i_{T^i(t)}(\tilde{x}^i(t), \tilde{u}^i(\cdot)) \tag{10}$$

Subject to:

$$\begin{aligned}
&\dot{x}^i(t) = f(x^i(t), u^i(t)) \; ; x^i \in \mathrm{X}^i, u^i \in U^i \\
&\dot{x}^j(t) = f(x^j(t), u^j(t)) \; ; x^j \in \mathrm{X}^j, u^j \in U^j, \; (i,j) \in E(t) \\
&g^{i,j}(x^i(t), u^i(t), x^j(t), u^j(t)) \le 0, (i,j) \in E(t) \\
&x^i(0) \in X^i_0, \; u^i(0) \in U^i_0 \\
&x^j(0) \in X^j_0, \; u^j(0) \in U^j_0, (i,j) \in E(t) \\
&x^i(T) \in X^i_f, u^i(T) \in U^i_f \; ; \\
&x^j(T) \in X^j_f, u^j(T) \in U^j_f, (i,j) \in E(t)
\end{aligned} \tag{11}$$

Where $J^i_{T^i(t)}(t)$ is defined in (6), X^i and U^i denotes the set of acceptable states and inputs respectively for i-th vehicle, X_0^i and X_f^i are the set of acceptable initial and final states of i-th vehicle respectively.

Assumption: Without losing the generality, it is assumed that the equilibrium point of the system (vehicle's dynamics) is origin. This assumption is not restrictive because in the cases where the equilibrium point is not origin a change in the variables can be used to transfer the equilibrium point to the origin.

Assumption: It is assumed that X_f^i is selected as small enough so that there exist a feedback control that drives $x^i(t+T^i(t))$ to the origin. If such a feedback control for $T^i(t)$ exist the optimization problem $P^i(t)$ is feasible.

The control action obtained from this optimization problem is implemented during the execution time $\delta^i(t) \in [t, t+T^i(t)]$ till the next update. Repeating this procedure online yields a close loop solution.

6.3.1 Variable Decentralized RHC Algorithm

In order to solve the above optimization problem, each vehicle needs to know current states, model, constraints and terminal region of itself and of its neighbours. Based on such information each vehicle computes the optimal inputs of itself and of its neighbours. The input to the neighbours will only be used to predict their trajectories and then discarded, while the first portion of the i-th optimal input of problem $P^i(t_k)$ will be implemented to the i-th vehicle during $[t_k, t_{k+1}]$ where $t_{k+1} = t_k + \delta^i(t_k)$, $k \in \square$ and $\delta^i(t_k)$ denotes the RHC sampling period and also called execution horizon. The following algorithm is proposed for the on-line implementation of the above optimization problem. The algorithm is formulated for the i-th vehicle; in fact, all vehicles run this algorithm during performing the mission simultaneously:

Variable Decentralized RHC Algorithm1:

1) Let *k=0*
2) Compute graph connection $E^i(t_k)$ according to (5).
3) Sample $x^i_{actual}(t_k)$.
4) Communicate $x^i_{actual}(t_k)$ to neighbour vehicles and receive $x^j_{actual}(t_k)$ for $(i,j) \in E^i(t_k)$.
5) Solve the decentralized optimization problem $P^i(t_k)$ and generate the control action for time interval $[t_k, t_k + T^i(t_k)]$.
6) Execute the control action for individual vehicle over the time interval $[t_k, t_{k+1}]$.

$$u^i(t) = u^{*i}_{t_k}(t);\ t \in [t_k, t_{k+1}] \qquad \textbf{(12)}$$

7) *k=k+1*. Goto step 2.

This algorithm is repeated till $x^i(t) = 0$ (assuming the origin is the equilibrium of the system). The 6th step of the algorithm allows the vehicles to share and update their information about the targets.

6.3.2 Variable Decentralized RHC Algorithm with Communication delay

Since in the decentralized RHC the neighbour vehicles are coupled through the cost function and constraints, each vehicle needs the updated information of its neighbours frequently at each sampling time (t_k) (execution time) to solve the optimization problem $P^i(t_k)$ and to generate its trajectory.

On the other hand, in reality the communicated information is subject to delay; therefore, even in the case of availability of communication, the current information of neighbour vehicles is not available for all vehicles, e.g. the current states of the neighbours of i-th vehicle is not available for i-th vehicle. To account for the communication delay, prediction must be done based on the most recent received information from neighbours. In cases where the communication delay is smaller than *execution horizon* the prediction problem is straightforward by communicating the control action and predicting the state of neighbours from their mathematical model.

However, in some applications in the real world the communication delay is bigger than execution horizon (or even prediction horizon). The communication delay comes from different sources; for instance, the limited band width of the communication facilities can induce a delay in the communicated data. Also, the slow sensors and/or the process of send and receive can cause a bigger communication delay. For another example one can point to the case of submarines; in the water the communicated signals can not travel as fast as the communicated signals in the air; hence, the communication in the water involves a bigger delay than the air. In these cases a new algorithm of decentralized RHC is required to account for communication delay. Suppose that the communication delay between i-th and j-th vehicle is denoted by τ^{ij} and $t_{k-d} \le t_k - \tau^{ij} < t_{k-d+1}$, $d \in \square$; also, $\tau^{ij} \le \min\{T^i(t_k), T^j(t_{k-d})\}$.

The following algorithm is proposed for the decentralized RHC algorithm. The algorithm is developed for the i^{th} vehicle; in fact, all vehicles run this algorithm during performing the mission simultaneously:

Variable Decentralized RHC Algorithm2:

1) Let $k=0$.
2) Compute graph connection $E^i(t_k)$ according to (5).
3) Sample $x^i_{actual}(t_k)$.
4) Communicate $x^i_{actual}(t_k)$ to neighbour vehicles and receive $x^j_{actual}(t_{k-d})$ where $(i,j)\in E^i(t_k)$ and goto step 7.
5) Solve the decentralized optimization problem $P^i(t_k)$ and generate the control action for time interval $[t_k, t_k + T^i(t_k)]$.
6) Execute the control action for individual vehicle over the time interval $[t_k, t_{k+1}]$.

   $$u^i(t) = u^{*i}_{t_k}(t);\ t\in[t_k, t_{k+1}] \qquad \textbf{(13)}$$
7) $k=k+1$. Goto step 2.

In step 5 of the above algorithm the state of neighbour vehicles at t_k is needed to solve the optimization problem p_i while this information is not available due to communication delay. The following procedure is proposed to predict $x^{j,i}(t_k)$ where $x^{j,i}_{t_0}(t)$ denotes the state of j-th vehicle at time step t, computed by i-th vehicle through solving the optimization problem $P^i(t_k)$:

At $t=t_k$ the following information from the neighbours are available:

- $x^{j,j}_{actual}(t_{k-d})$

- $u^{j,j}_{t_{k-d}}(s);\ s\in\left[t_{k-d}, t_{k-d}+T^j(t_{k-d})\right]$

Hence, solving the following ordinary differential equation yields the solution:

$$\dot{x}^j = f(x^j, u^j); \begin{cases} x^{j,i}(t_{k-d}) = x^{j,j}_{actual}(t_{k-d}) \\ u^{j,i}(s) = u^{j,j}_{actual}(s); s \in [t_{k-d}, t_k] \end{cases} \quad (14$$

Remark: If $\tau^{ij} \le \delta^i(t_{k-d}) \le \delta^j(t_{k-d})$ then $u^{j,i}(s) = u^{j,j}(s)$; therefore, if there is no model uncertainty $x^{j,i}(t_k) = x^{j,j}(t_k)$. Generally, if $T^i(t_{k-d}) - \delta^i(t_{k-d}) \ge \tau^{ij} \ge \delta^i(t_{k-d})$ then $u^{j,i}(s) \ne u^{j,j}(s)$ because i-th vehicle uses some portions of $u^{j,j}(s); s \in \left[t_{k-d+1}, t_{k-d} + T^i(t_{k-d})\right]$ from the previous optimization problem while j-th vehicle uses the updated control action. Therefore, even in the case of no model uncertainty it may lead to a mismatch between $x^{j,i}(t_k)$ and $x^{j,j}(t_k)$ which induces a mismatch between the trajectories that $P^i(t_k)$ and $P^j(t_k)$ generate for the j-th vehicle during $[t_k, t_k + T^i(t_k)]$, this may lead to infeasibility and instability of $P^i(t_k)$, the following sections will address this issue and finds a bounds on the control action and mismatch to ensure the feasibility and stability of the decentralized RHC problem.

6.4 Stability condition of Decentralized RHC for continuous time systems

In the decentralized RHC formulated here, the vehicles use a model of the neighbour vehicles to predict their trajectory. The predicted neighbours' trajectories will be used to meet the sufficient condition for stability of the whole group. In addition, the information communicated between the vehicles is used after each sampling time to make the predicted solutions feasible. The following proposition gives the sufficient condition for the stability of the decentralized RHC problem $P^i(t_k)$.

Proposition 1: Suppose that matrix penalties Q and R are symmetric, positive and full rank. Also, assume the sets of states and inputs contain the origin. Then, the following inequality is the sufficient condition for the asymptotic stability of the decentralized RHC problem $P^i(t_k)$ at the origin:

$$\varepsilon^i(t_k) \le \kappa^i(t_k) \tag{15}$$

where ε^i is the summation of the prediction mismatches over the prediction horizon:

$$\varepsilon^i(t_k) = \sum_{j|(i,j)\in E^i(t_k)} \left[\int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| x_{t_k}^{j,j}(t) - x_{t_k}^{j,i}(t) \right\|_Q^2 dt + \int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| u_{t_k}^{j,j}(t) - u_{t_k}^{j,i}(t) \right\|_Q^2 dt \right] \tag{16}$$

Also:

$$\kappa^i(t_k) = \int_{t_k}^{t_k+\delta^i(t_k)} \left(\left\| x_{t_k}^{i,i}(t) \right\|_Q^2 + \left\| u_{t_k}^{i,i}(t) \right\|_R^2 \right) dt + \sum_{j|(i,j)\in E^i(t_k)} \left[\int_{t_k}^{t_k+\delta^i(t_k)} \left(\left\| x_{t_k}^{j,i}(t) \right\|_Q^2 + \left\| u_{t_k}^{j,i}(t) \right\|_R^2 \right) dt \right] \tag{17}$$

Proof: Without losing the generality consider the case of two vehicles: (*i=1, j=2*); then:

$$\begin{aligned} J_T^{1*}(\tilde{x}^{*1}(t_{k+1}), \tilde{u}^{*1}(\cdot)) = & \int_{t_{k+1}}^{t_{k+1}+T^1(t_k)} \left(\left\| x_{t_{k+1}}^{1,1}(t) \right\|_Q^2 + \left\| u_{t_{k+1}}^{1,1}(t) \right\|_R^2 \right) dt + \left\| x_{t_{k+1}}^{1,1}(t_{k+1}+T) \right\|_P^2 \\ & + \int_{t_{k+1}}^{t_{k+1}+T^1(t_k)} \left(\left\| x_{t_{k+1}}^{2,1}(t) \right\|_Q^2 + \left\| u_{t_{k+1}}^{2,1}(t) \right\|_R^2 \right) dt + \left\| x_{t_{k+1}}^{2,1}(t_{k+1}+T) \right\|_P^2 \end{aligned} \tag{18}$$

Also:

$$\begin{aligned} J_T^{1*}(\tilde{x}^{*i}(t_k), \tilde{u}^{*i}(\cdot)) = & \int_{t_k}^{t_k+T} \left(\left\| x_{t_k}^{1,1}(t) \right\|_Q^2 + \left\| u_{t_k}^{1,1}(t) \right\|_R^2 \right) dt + \left\| x_{t_k}^{1,1}(t_k+T) \right\|_P^2 \\ & + \int_{t_k}^{t_k+T} \left(\left\| x_{t_k}^{2,1}(t) \right\|_Q^2 + \left\| u_{t_k}^{2,1}(t) \right\|_R^2 \right) dt + \left\| x_{t_k}^{2,1}(t_k+T) \right\|_P^2 \end{aligned} \tag{19}$$

Also consider the optimization problem $P^1(t_k)$ with the following optimal solutions:

$$\begin{aligned} & u_{t_k}^{1,1*}(s);\ s \in [t_k, t_k + T^1(t_k)] \\ & u_{t_k}^{2,1*}(s);\ s \in [t_k, t_k + T^1(t_k)] \end{aligned} \tag{20}$$

$u_{t_k}^{2,1}(t_{k+1})$ and $u_{t_k}^{2,2}(t_{k+1})$ are calculated from two different optimization problems $P^1(t_k)$ and $P^2(t_k)$ respectively; hence, they are not necessarily the same and it means the following shifted optimization solutions may not be feasible:

$$\begin{aligned} u_{t_{k+1}}^{1,1}(s) &= \begin{cases} u_{t_k}^{1,1*}(s) ; & s \in [t_{k+1},\ t_{k+1} + T^1(t_{k+1}) - \delta^1(t_{k+1})] \\ 0 & ; s \in [t_{k+1} + T - \delta^1(t_{k+1}), t_{k+1} + T^1(t_{k+1})] \end{cases} \\ u_{t_{k-1}}^{2,1}(s) &= \begin{cases} u_{t_k}^{2,1*}(s) ; & s \in [t_{k+1},\ t_{k+1} + T^1(t_{k+1}) - \delta^1(t_{k+1})] \\ 0 & ; s \in [t_{k+1} + T^1(t_{k+1}) - \delta^1(t_{k+1}), t_{k+1} + T^1(t_{k+1})] \end{cases} \end{aligned} \tag{21}$$

Hence, we construct the following feasible control trajectory:

$$\begin{aligned} u_{t_{k-1}}^{1,1}(s) &= \begin{cases} u_{t_k}^{1,1*}(s) ; & s \in [t_{k+1},\ t_{k+1} + T^1(t_k) - \delta^1(t_k)] \\ 0 & ; s \in [t_{k+1} + T^1(t_k) - \delta^1(t_k), t_{k+1} + T^1(t_{k+1})] \end{cases} \\ u_{t_{k-1}}^{2,1}(s) &= \begin{cases} u_{t_k}^{2,2*}(s) ; & s \in [t_{k+1},\ t_{k+1} + T^2(t_k) - \delta^2(t_k)] \\ 0 & ; s \in [t_{k+1} + T^2(t_k) - \delta^2(t_k), t_{k+1} + T^1(t_k) - \delta^1(t_k)] \\ 0 & ; s \in [t_{k+1} + T^1(t_k) - \delta^1(t_k), t_{k+1} + T^1(t_{k+1})] \end{cases} \end{aligned} \tag{22}$$

Since the dynamics of the vehicles are not coupled it is possible to construct this feasible solution. This feasible control trajectory will be used to calculate the cost function (23) which is not optimal necessarily:

$$\begin{aligned} J_{T^1(t_{k+1})}^{1}(\tilde{x}^1(t_{k+1}), \tilde{u}^1(\cdot)) = \int_{t_{k+1}}^{t_{k+1}+T^1(t_{k+1})} (\|x_{t_k}^{1,1}(t)\|_Q^2 + \|u_{t_k}^{1,1}(t)\|_R^2)dt + \|x_{t_k}^{1,1}(t_{k+1} + T^1(t_{k+1}))\|_P^2 \\ + \int_{t_{k+1}}^{t_{k+1}+T^1(t_{k+1})} (\|x_{t_k}^{2,1}(t)\|_Q^2 + \|u_{t_k}^{2,1}(t)\|_R^2)dt + \|x_{t_k}^{2,1}(t_{k+1} + T^1(t_{k+1}))\|_P^2 \end{aligned} \tag{23}$$

Also, since we assumed the dynamics are decoupled, $u_{t_{k+1}}^{1,1}$ and $u_{t_{k+1}}^{2,2}$ are feasible at time $t = t_{k+1}$ for problem $P^1(t_k)$. The figure below shows schematically the three cost function which will be used to prove the stability:

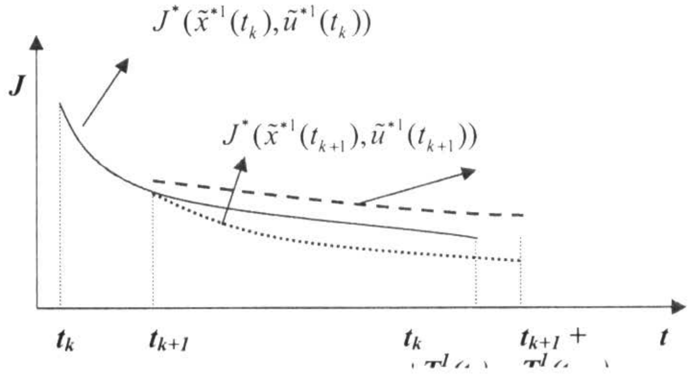

$J^*(\tilde{x}^{*1}(t_k), \tilde{u}^{*1}(t_k))$
J
$J^*(\tilde{x}^{*1}(t_{k+1}), \tilde{u}^{*1}(t_{k+1}))$
t_k
t_{k+1}
t_k
t_{k+1} +
t

Fig. 5: Schematic representation of cost functions used to prove the stability

Hence, from the optimality property we have:

$$
\begin{aligned}
& J^{1*}_{T^1(t_{k-1})}(\tilde{x}^{*1}(t_{k+1}),\tilde{u}^{*1}(t_{k+1})) \le J^{1}_{T^1(t_{k+1})}(\tilde{x}^{1}(t_{k+1}),\tilde{u}^{1}(t_{k+1})) \\
& = J^{1*}_{T^1(t_k)}(\tilde{x}^{*1}(t_k),\tilde{u}^{*1}(t_k)) \\
& - \int_{t_k}^{t_{k+1}} (\|x^{1,1}_{t_k}(t)\|^2_Q + \|u^{1,1}_{t_k}(t)\|^2_R)dt - \|x^{1,1}_{t_k}(t_k + T^1(t_k))\|^2_P \\
& - \int_{t_k}^{t_{k+1}} (\|x^{2,1}_{t_k}(t)\|^2_Q + \|u^{2,1}_{t_k}(t)\|^2_R)dt - \|x^{2,1}_{t_k}(t_k + T^1(t_k))\|^2_P \\
& + \int_{t_{k+1}}^{t_k+T^1(t_k)} (\|x^{2,2}_{t_k}(t)\|^2_Q - \|x^{2,1}_{t_k}(t)\|^2_Q)dt + \int_{t_{k+1}}^{t_k+T^1(t_k)} (\|u^{2,2}_{t_k}(t)\|^2_Q - \|u^{2,1}_{t_k}(t)\|^2_Q)dt \\
& + \int_{t_k+T^1(t_k)}^{t_{k+1}+T^1(t_{k+1})} (\|x^{1,1}_{t_k}(t)\|^2_Q + \|x^{2,1}_{t_k}(t)\|^2_Q)dt \\
& + \|x^{1,1}_{t_k}(t_{k+1} + T^1(t_{k+1}))\|^2_P + \|x^{2,1}_{t_k}(t_{k+1} + T^1(t_{k+1}))\|^2_P
\end{aligned}
\tag{24}
$$

If matrix penalty P is calculated from the control Lyapanov function (CLF) of the linearized model of (9), then the final cost $\|x(t_k+T)\|^2_P$ is a bound on the finite optimal cost function[19]; in fact:

$$
\begin{aligned}
& \|x^{1,1}_{t_k}(t_k + T^1(t_k))\|^2_P + \|x^{2,1}_{t_k}(t_k + T^1(t_k))\|^2_P \ge \int_{t_k}^{t_k+T^1(t_k)} (\|x^{1,1}_{t_k}(t)\|^2_Q + \|x^{2,1}_{t_k}(t)\|^2_Q)dt \\
& \|x^{1,1}_{t_k}(t_{k+1} + T^1(t_{k+1}))\|^2_P + \|x^{2,1}_{t_{k-1}}(t_k + T^1(t_{k+1}))\|^2_P \ge \int_{t_{k+1}}^{t_{k+1}+T^1(t_{k+1})} (\|x^{1,1}_{t_k}(t)\|^2_Q + \|x^{2,1}_{t_k}(t)\|^2_Q)dt
\end{aligned}
\tag{25}
$$

Hence:

$$
\begin{aligned}
& \|x^{1,1}_{t_{k-1}}(t_{k+1} + T^1(t_{k+1}))\|^2_P + \|x^{2,1}_{t_{k-1}}(t_{k+1} + T^1(t_{k+1}))\|^2_P - \|x^{1,1}_{t_k}(t_k + T^1(t_k))\|^2_P - \|x^{2,1}_{t_k}(t_k + T^1(t_k))\|^2_P + \\
& \int_{t_k+T^1(t_k)}^{t_{k+1}+T^1(t_{k+1})} (\|x^{1,1}_{t_k}(t)\|^2_Q + \|x^{2,1}_{t_k}(t)\|^2_Q)dt \le 0
\end{aligned}
\tag{26}
$$

Using (26) in (24) and using triangle inequality for the weighted norms yields:

$$\begin{aligned}
&J^{1*}_{T^1(t_{k-1})}(\tilde{x}^{*1}(t_{k+1}),\tilde{u}^{*1}(t_{k+1})) - J^{1*}_{T^1(t_k)}(\tilde{x}^{*1}(t_k),\tilde{u}^{*1}(t_k)) \le \\
&+\int_{t_{k+1}}^{t_k+T^1(t_k)} \left\|x^{2,2}_{t_k}(t) - x^{2,1}_{t_k}(t)\right\|_Q^2 dt + \int_{t_{k+1}}^{t_k+T^1(t_k)} \left\|u^{2,2}_{t_k}(t) - u^{2,1}_{t_k}(t)\right\|_Q^2 dt \\
&-\int_{t_k}^{t_{k+1}} (\left\|x^{1,1}_{t_k}(t)\right\|_Q^2 + \left\|u^{1,1}_{t_k}(t)\right\|_R^2) dt - \int_{t_k}^{t_{k+1}} (\left\|x^{2,1}_{t_k}(t)\right\|_Q^2 + \left\|u^{2,1}_{t_k}(t)\right\|_R^2) dt
\end{aligned} \tag{27}$$

According to Lyapanov stability analysis for the stability of the decentralized RHC, the right hand side of inequality (27) must be negative which yields the following stability condition for the case of two vehicles:

$$\begin{aligned}
&\int_{t_{k+1}}^{t_k+T^1(t_k)} \left\|x^{2,2}_{t_k}(t) - x^{2,1}_{t_k}(t)\right\|_Q^2 dt + \int_{t_{k+1}}^{t_k+T^1(t_k)} \left\|u^{2,2}_{t_k}(t) - u^{2,1}_{t_k}(t)\right\|_Q^2 dt \\
&\le \int_{t_k}^{t_{k+1}} (\left\|x^{1,1}_{t_k}(t)\right\|_Q^2 + \left\|u^{1,1}_{t_k}(t)\right\|_R^2) dt + \int_{t_k}^{t_{k+1}} (\left\|x^{2,1}_{t_k}(t)\right\|_Q^2 + \left\|u^{2,1}_{t_k}(t)\right\|_R^2) dt
\end{aligned} \tag{28}$$

A same analysis for the general case with N_n^i number of neighbours for i-th vehicle yields the general stability condition (15). Stability condition (15) implies that if the prediction mismatch is bounded then the stability is ensured.

Remark: The right hand side of (27) can be used to define the stability margin for the i-th vehicle in general case as follows:

$$\begin{aligned}
&Stability\ Margin = \\
&-\sum_{j|(i,j)\in E^i(t_k)} \int_{t_{k+1}}^{t_k+T^i(t_k)} \left\|x^{j,j}_{t_k}(t) - x^{j,i}_{t_k}(t)\right\|_Q^2 dt + \int_{t_{k-1}}^{t_k+T^i(t_k)} \left\|u^{j,j}_{t_k}(t) - u^{j,i}_{t_k}(t)\right\|_Q^2 dt \\
&+\int_{t_k}^{t_k+\delta^i(t_k)} (\left\|x^{i,i}_{t_k}(t)\right\|_Q^2 + \left\|u^{i,i}_{t_k}(t)\right\|_R^2) dt + \int_{t_k}^{t_k+\delta^i(t_k)} (\left\|x^{j,i}_{t_k}(t)\right\|_Q^2 + \left\|u^{j,i}_{t_k}(t)\right\|_R^2) dt \\
&= -\varepsilon^i(t_k) + \kappa^i(t_k)
\end{aligned} \tag{29}$$

The bigger stability margin the more stable system. This definition of stability margin will be used in the future sections to investigate the effectiveness of each method for enhancing the stability.

6.5 Improved algorithms for decentralized RHC

In this section the role of variable communication topology, variable prediction horizon and variable sampling time for stability enhancement is investigated, and finally an algorithm is proposed which uses the results of this section for further enhancement of stability of decentralized RHC.

Beforehand, it is important to find out how the communication can be performed to enhance the stability. According to (29), for a larger stability margin the integration of mismatch between the predicted and actual trajectories of the neighbours over the prediction horizon must be as small as possible. Hence, in order to enhance the stability of the system ε^i should be reduced as much as possible and the communication must be appropriately performed in this way. In fact, the communication should be used in such a way that the predicted trajectory of the neighbour vehicles don't deviate too far from the actual values. Therefore, in this section, the communication techniques are utilized in such a way to reduce the mismatch between the predicted and actual trajectories of neighbour vehicles.

6.5.1 Variable Communication topology

By the expression "variable communication topology" we mean that the number of the neighbours of each vehicle may change during performing the mission; in fact, the set of neighbour vehicles of each vehicle is time varying. The variation in the number of neighbour vehicles of each vehicle is carried out by two communication techniques: 1- changing the communication radius and 2- multi-hopping communication. These two techniques are introduced and discussed in this section.

6.5.1.1 Expanding the communication radius:

The communication radius defines a region around each individual vehicle where the individual vehicle is able to communicate with vehicles which are inside this region. Figure **1** shows this region for a vehicle by a circle, the dashed circles show the expanded communication radius and it shows that with a larger communication radius the vehicle is able to communicate with more other vehicles. Hence, the larger communication radius allows more neighbours for each vehicle and this drives the local solutions of the decentralized RHC to be closer to the global centralized solution which is unique for all vehicles. This leads to smaller mismatch between predicted and actual trajectories of neighbours, and according to stability margin (29) this will increase the stability margin of the decentralized RHC. Therefore, generally speaking one can say that a larger communication radius can improve the stability of the decentralized RHC.

However, in the case of larger communication radius the communication delay may deteriorate the performance of the system. Also, for more number of neighbours, the scale of the decentralized RHC problem is larger and this leads to the higher computation burden. Hence, a trade-off between these issues provides an optimum communication radius.

6.5.1.2 Multi-hopping communication:

Another potential way for changing the number of vehicles is multi-hopping communication. From a communication point of view, each node (vehicle) on the graph is a hop that can receive the information from the neighbour nodes and transfer the information to them. The idea is to hop (transfer) the information from one node of the graph to another until the information arrives at the destination[22]. In fact, the information is communicated between two nodes which are not necessarily neighbours.

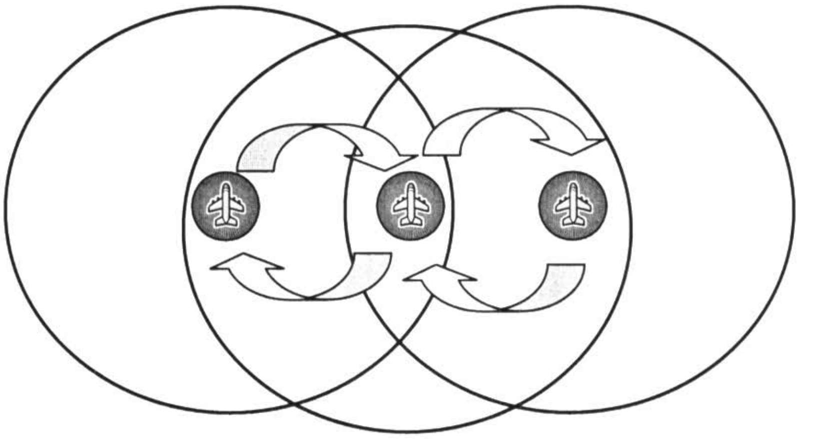


Fig. 6: Multi-hopping communication and information flow between two non-neighbour vehicles

Multi-hopping communication enables each vehicle to communicate with more vehicles through its neighbours. It is true to say that the i-th vehicle has access to all vehicles which its neighbours have access to them. Figure 6 shows how two non-neighbouring vehicles can communicate through a common neighbour. The arrows show the flow of information between two non-neighbours vehicles communicating through a common neighbour.

Using multi-hoping communication, more vehicles are able to communicate with each other since they are all connected through multi-hopping. Hence, this will make the set of neighbours larger and consequently this allows the local solutions of the decentralized RHC to be closer to the global centralized solution which is unique for all vehicles. This leads to smaller mismatch between predicted and actual trajectories of neighbours and hence according to stability margin (29) this will increase the stability margin of the decentralized RHC. Therefore, theoretically, one can say that the multi-hopping can improve the stability of the decentralized RHC. However, practically multi-hopping induces delays and noise that will adversely affect the control algorithm[23]. This delay is different for each neighbour and it is seen in the literatures that the effect of the varying delay is more deteriorative than the constant delay[24]. Also, the effect of the higher computation effort must be considered like the case of larger communication radius.

6.5.2 Variable Prediction Horizon

The effect of the variable prediction horizon is investigated in this section. We will find some bounds on the prediction horizon so that the stability is achieved. Since each vehicle uses a model of the neighbour to predict the trajectory of neighbour vehicles, to investigate the effect of variable prediction horizon on the stability of decentralized RHC, two cases will be studied: 1) model of neighbour vehicle without uncertainty 2) model of neighbour vehicle with model uncertainty. Beforehand, we make the following assumption:

Assumption: we assume that the dynamics of vehicles is *Lipchitz continuous* with *Lipchitz constants* L_u and L_x as follows:

$$\left\| f(x_1, u_1) - f(x_2, u_2) \right\| \le L_x \left\| x_1 - x_2 \right\| + L_u \left\| u_1 - u_2 \right\| \tag{30}$$

6.5.2.1 No uncertainty in the model of neighbour vehicle

Suppose that μ is the mismatch between the predicted and actual control action of neighbour j:

$$\left\| u_{t_k}^{j,j}(t) - u_{t_k}^{j,i}(t) \right\| \le \mu \qquad ; t \in \left[t_k , t_k + T \right] \tag{31}$$

Hence, from *Lipchitz continuous* assumption, the following hold:

$$\left\| f(x_{t_k}^{j,j}, u_{t_k}^{j,j}(t)) - f(x_{t_k}^{j,j}, u_{t_k}^{j,i}(t)) \right\| \le \mu L_u \tag{32}$$

Theorem 1: *Sufficient condition for stability of the decentralized RHC architecture is that every i-th vehicle satisfies the following inequality at each time step t_k:*

$$\sum_{j|(i,j)\in E^i(t_k)} \left[\frac{\left\| x_{t_k}^{j,j}(t_k) - x_{t_k}^{j,i}(t_k) \right\|}{L_x} \left(e^{L_x T^i(t_k)} - e^{L_x \delta^i(t_k)} \right) + \frac{\mu L_u}{L_x} \left(\frac{1}{L_x} (e^{L_x T^i(t_k)} - e^{L_x \delta^i(t_k)}) - (T^i(t_k) - \delta^i(t_k)) \right) \right] \lambda_{\max}(Q)$$
$$+ N_n^i \mu (T^i(t_k) - \delta^i(t_k)) \lambda_{\max}(R) \le \kappa^i(t_k) \tag{33}$$

where, μ is defined in (31), L is lipchitz constant of vehicles dynamics, $\|\cdot\|$ denotes the induced weighted norm of matrix, N_n^i is the number of neighbours of i-th vehicle and λ_{max} denotes the maximum eigenvalue of matrix.

Proof: For simplicity, we first consider the case of two vehicles (*Nv=2*), hence:

$$\varepsilon^i = \int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| x_{t_k}^{j,j}(t) - x_{t_k}^{j,i}(t) \right\|_Q^2 dt + \int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| u_{t_k}^{j,j}(t) - u_{t_k}^{j,i}(t) \right\|_Q^2 dt \tag{34}$$

Where, *j* is the neighbour of *i*, and (*i=1, j=2*). Also, from (31) we have:

$$\int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| u_{t_k}^{j,j}(t) - u_{t_k}^{j,i}(t) \right\| dt \le \mu (T^i(t_k) - \delta^i(t_k)) \tag{35}$$

Using Bellman-Gronwall lemma for mismatch (32) and taking integration yields:

$$\int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| x_{t_k}^{j,j}(t) - x_{t_k}^{j,i}(t) \right\| dt \le \int_{t_{k+1}}^{t_k+T^i(t_k)} \left[\left\| x_{t_k}^{j,j}(t_k) - x_{t_k}^{j,i}(t_k) \right\| e^{L_x(t-t_k)} + \frac{\mu L_u}{L_x}(e^{L_x(t-t_k)} - 1) \right] dt \tag{36}$$

Where, L_x is the Lipchitz constant for system (9) with respect to x. Some mathematical operations yield:

$$\begin{gathered} \int_{t_{k+1}}^{t_k+T^i(t_k)} \left\| x_{t_k}^{j,j}(t) - x_{t_k}^{j,i}(t) \right\| dt \le \frac{\left\| x_{t_k}^{j,j}(t_k) - x_{t_k}^{j,i}(t_k) \right\|}{L_x} \left[e^{L_x T^i(t_k)} - e^{L_x \delta^i(t_k)} \right] + \\ \frac{\mu L_u}{L_x} \left[\frac{1}{L_x} (e^{L_x T^i(t_k)} - e^{L_x \delta^i(t_k)}) - (T^i(t_k) - \delta^i(t_k)) \right] \end{gathered} \tag{37}$$

Putting (35) and (37) into (34) yields to following bound on the ε^i and Using this bound in (15) yields:

$$\begin{gathered} \varepsilon^i \le \left[\frac{\left\| x_{t_k}^{j,j}(t_k) - x_{t_k}^{j,i}(t_k) \right\|}{L_x} \left(e^{L_x T^i(t_k)} - e^{L_x \delta^i(t_k)} \right) + \frac{\mu L_u}{L_x} \left(\frac{1}{L_x} (e^{L_x T^i(t_k)} - e^{L_x \delta^i(t_k)}) - (T^i(t_k) - \delta^i(t_k)) \right) \right] \lambda_{max}(Q) \\ + \mu (T^i(t_k) - \delta^i(t_k)) \lambda_{max}(R) \le k^i \end{gathered} \tag{38}$$

The proof for the general case with *Nv* vehicles follows along the lines of the case of two vehicles and yields the stability condition (33) that defines a bound on the *prediction horizon T*. Since the left hand side of (33) is a strictly increasing function of *prediction* *horizon T,* there is a unique solution for inequality (33). Evidently this solution comes from the communication limitation and hence is called T_{Comu}.

6.5.2.2 Model of neighbour vehicle with model uncertainty

Assume that there is the following bound on the model uncertainty:

$$\left\| f(x^{j}) - f_{actual}(x^{j}) \right\| \leq \beta \qquad \textbf{(39)}$$

Theorem 2: *Sufficient condition for stability of the decentralized RHC architecture is that every i-th vehicle satisfies the following inequality:*

$$\sum_{j|(i,j)\in E^{i}(t_k)} \left[\frac{\left\| x_{t_k}^{j,j}(t_k) - x_{t_k}^{j,i}(t_k) \right\|}{L_x} \left(e^{L_x T^{i}(t_k)} - e^{L_x \delta^{i}(t_k)} \right) + \frac{(\mu L_u + \beta)}{L_x} \left(\frac{1}{L_x} (e^{L_x T^{i}(t_k)} - e^{L_x \delta^{i}(t_k)}) - (T^{i}(t_k) - \delta^{i}(t_k)) \right) \right] \lambda_{\max}(Q) \qquad \textbf{(40)}$$
$$+ N_n^{i} \mu (T^{i}(t_k) - \delta^{i}(t_k)) \lambda_{\max}(R) \leq \kappa^{i}(t_k)$$

Where, μ *is defined in (31),* β *is model uncertainty of neighbour vehicles, L is Lipchitz constant of vehicles dynamics,* $\|\cdot\|$ *denotes the induced weighted norm of matrix,* N_n^{i} *is the number of neighbours of i-th vehicle and* λ_{max} *denotes the maximum eigenvalue of matrix.*

Proof: An analysis similar to theorem 1 provides the proof for this theorem.

The inequalities (33) and (40) suggest that for stability of the decentralized RHC the following must hold:

$$T \leq T_{comu} \qquad \textbf{(41)}$$

In fact, the prediction horizon should be smaller than T_{Comu}. However, we should consider that normally for RHC the prediction horizon should be selected as large as possible for better performance. Hence, selection of *T* is a trade off between stability and performance.

6.5.3 Variable Communication Rate

The communication sampling time is equal to the *execution time* of the RHC. Also, it is well known that for stability of RHC the *execution time* must be smaller than the prediction horizon [19]; hence, it is concluded that for stability the communication rate must be smaller than the prediction horizon coming from (33) and (40). On the other hand, it is required that each vehicle has enough time to transfer its updated information to its neighbours. In fact, the amount of communicated information mustn't exceed the bandwidth of the available communication facilities at each sampling time; hence, the following communication constraint must be respected:

$$\sum \frac{K_i(t)}{\delta_c^i(t)} \le B \tag{42}$$

where, $\delta_c^i(t)$ is the sampling period of the i-th vehicle at time t, K_i is the amount of i-th package (bits/sample) and B (bits/s) is the bandwidth of the available communication channel. This will impose a lower bound on the communication rate. Thus:

$$\delta_c^i(t_k) \le \delta^i(t_k) << T^i(t_k) \tag{43}$$

The following theorem suggests that to reduce the deteriorative effects of communication delay on the stability, the communication should be performed with a faster rate. Without losing the generality, the analysis is carried out for linear systems while the results can be generalized to a wider class of systems (the nonlinear systems can be approximated by a linear system)

Theorem 3: *Consider the optimization problem 1 with the following linear differential equation describing the dynamics of i-th vehicle:*

$$\begin{aligned} \dot{x}^i(t) &= Ax^i(t) + Bu^i(t) \\ x^i(t_0) &= x^i{}_0 \in X_0^i \end{aligned} \tag{44}$$

And also suppose that the following provides the concatenated dynamics of all vehicles in the set of neighbours of i-th vehicle:

$$\begin{aligned}\dot{\tilde{x}}^i(t) &= \tilde{A}^i \tilde{x}^i(t) + \tilde{B}^i \tilde{u}^i(t)\\ \tilde{x}^i(t_0) &= \tilde{x}_0^i\end{aligned} \tag{45}$$

Then, the following holds for each vehicle at each time step t_k:

$$\left\| x_{t_{k-1}}^{j,j}(t_k) - x_{t_{k-1}}^{j,i}(t_k) \right\| \le \alpha_1 e^{\lambda_1 \delta^i(t_k)} \left\| x^{j,j}(t_{k-d}) \right\| \tag{46}$$

Where α_1 and λ_1 are positive numbers.

Proof: the *Hamiltonian* for the optimization problem 1 is written as follows[25]:

$$H = \|\tilde{x}\|_{\tilde{Q}}^2 + \|\tilde{u}\|_{\tilde{R}}^2 + q^T(\tilde{A}^i \tilde{x}^i + \tilde{B}^i \tilde{u}^i) \tag{47}$$

Where q defines the vector of costaes. Hence, the necessary condition for the optimality is:

$$\nabla_{\tilde{u}} H = 0 \Rightarrow \tilde{u}(t) = -(\tilde{R} + \tilde{R}^T)^{-1} \tilde{B}^T \tilde{P} = -\frac{1}{2} \tilde{R}^{-1} \tilde{B}^T \tilde{P} \tag{48}$$

Also, we have:

$$\dot{q} = -\nabla_{\tilde{x}} H \tag{49}$$

Combining this with system dynamics yields:

$$\begin{bmatrix} \dot{\tilde{x}} \\ \dot{q} \end{bmatrix} = \begin{bmatrix} \tilde{A} & -\frac{1}{2} \tilde{B} \tilde{R}^{-1} \tilde{B}^T \\ -2\tilde{Q} & -\tilde{A} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ q \end{bmatrix} \tag{50}$$

And for the boundary conditions we have:

$$q(T^i(t_k)) = \nabla_x \|\tilde{x}\|_p^2 \Big|_{T^i(t_k)} = \tilde{x}(T^i(t_k)) \tag{51}$$

Solving the system of differential equations (50) yields the following solutions:

$$\begin{aligned}\tilde{u}(t) &= -\frac{1}{2} \tilde{R}(t_k)^{-1} \tilde{B}(t_k)^T \varphi_{t_k}^u(t) \tilde{x}(t_k); t \in [t_k, t_k + T^i(t_k)]\\ \tilde{x}(t) &= \varphi_{t_k}^x(t) \tilde{x}(t_k); t \in [t_k, t_k + T^i(t_k)]\end{aligned} \tag{52}$$

Where,

$$\begin{aligned}\varphi_{t_k}^u(t) &= \varphi_3(t) + \varphi_4(t).S_{t_k}\\ \varphi_{t_k}^x(t) &= \varphi_1(t) + \varphi_2(t).S_{t_k}\end{aligned} \tag{53}$$

Where,

$$S_{t_k} = -\frac{1}{2}\tilde{R}^{-1}\tilde{B}^T[\varphi_4(T^i(t_k)) - 2\tilde{P}\varphi_2(T^i(t_k))]^{-1}[2\tilde{P}\varphi_1(T^i(t_k)) - \varphi_3(T^i(t_k))] \qquad \textbf{(54)}$$

And $\varphi_1, \varphi_2, \varphi_3$ and φ_4 are the components of the transition matrix of system (50):

$$\varphi(t, t_0) = \begin{bmatrix} \varphi_1 & \varphi_2 \\ \varphi_3 & \varphi_4 \end{bmatrix} \qquad \textbf{(55)}$$

In [21] it is proven that the following bounds can be defined on the transition matrix components:

$$\begin{aligned} \|\varphi_u(t)\| &\le \alpha_u e^{\lambda_u t}; \alpha_u, \lambda_u \ge 0 \\ \|\varphi_x(t)\| &\le \alpha_x e^{\lambda_x t}; \alpha_x, \lambda_x \ge 0 \end{aligned} \qquad \textbf{(56)}$$

If we separate the inputs and states evolution equation for the j-th vehicle and shift that in time by *d* steps, we will have:

$$\begin{aligned} u^j_{t_{k-d}}(t) &= -\frac{1}{2}(R^j)^{-1}(B^j)^{T^i(t_k)}\varphi^u_{t_{k-d}}(t)x(t_{k-d}); t \in [t_{k-d}, t_{k-d} + T^i(t_k)] \\ \tilde{x}^j_{t_{k-d}}(t) &= \varphi^x_{t_{k-d}}(t)\tilde{x}^j_{t_{k-d}}(t_{k-d}); t \in [t_{k-d}, t_{k-d} + T^i(t_k)] \end{aligned} \qquad \textbf{(57)}$$

Since $\varphi^u_{t_{k-d}}$ and $\varphi^x_{t_{k-d}}$ obtained from the graph topology $E^j_{t_{k-d}}$ the control action above is a function of graph topology. In the decentralized RHC the j-th vehicle transmit this control action to its neighbour vehicle in order to the neighbours use this control action for prediction of $x^{j,j}_{t_k}$; in fact, the i-th vehicle uses this control action over the horizon $[t_{k-d}, t_k]$ while the j-th vehicle use this control action during $[t_{k-d}, t_{k-d+1}]$and at the next time step the j-th vehicle uses the updated control trajectory $u^{j,j}_{t_{k-d+1}}(t)$; hence, this mismatch between what j-th vehicle plans and what i-th vehicle assumes about that induces a delay induces a mismatch between $x^{j,j}_{t_k}$ and $x^{j,i}_{t_k}$. Therefore, since $x^{j,j}_{t_k}$ and $x^{j,i}_{t_k}$ are used to solve optimization problems $P^j(t_k)$ and $P^i(t_k)$ it is required to find the mismatch between $x^{j,j}_{t_k}$ and $x^{j,i}_{t_k}$ as follows:

$$\begin{aligned} x^{j,j}_{t_{k\,1}}(t_k) = \varphi^x_{t_{k\,1}}(t_k)x^{j,j}_{t_{k\,2}}(t_{k-1}) = \varphi^x_{t_{k\,1}}(t_k)\varphi^x_{t_{k\,2}}(t_{k-1})x^{j,j}_{t_{k\,3}}(t_{k-2}) = \\ \varphi^x_{t_{k\,1}}(t_k)\varphi^x_{t_{k\,2}}(t_{k-1})\ldots\varphi^x_{t_{k\,d}}(t_{k-d+1})x^{j,j}(t_{k-d}) \end{aligned} \qquad \textbf{(58)}$$

Also:

$$x_{t_{k-d}}^{j,i}(t_k) = \varphi_{t_{k-d}}^{x}(t_k)x^{j,i}(t_{k-d}) \tag{59}$$

Since $x^{j,j}(t_{k-d})$ is transmitted to i-th vehicle and it is received by i-th vehicle at time t_k, then $x^{j,j}(t_{k-d}) = x^{j,i}(t_{k-d})$. Subtraction (58) and (59), taking norm and using inequality (56) yield:

$$\begin{aligned}\left\|x_{t_{k-1}}^{j,j}(t_k) - x_{t_{k-1}}^{j,i}(t_k)\right\| &= \left\|\varphi_{t_{k-1}}^{x}(t_k)\varphi_{t_{k-2}}^{x}(t_{k-1})\dots\varphi_{t_{k-d}}^{x}(t_{k-d+1}) - \varphi_{t_{k-d}}^{x}(t_k)\right\|\left\|x^{j,j}(t_{k-d})\right\| \\ &\leq \alpha_1 e^{\lambda_1\delta}\left\|x^{j,j}(t_{k-d})\right\|\end{aligned} \tag{60}$$

Where α_1 and λ_1 are positive numbers.

Remark: inequality (60) says that a larger sampling time $\delta^i(t_k)$, will increase the mismatch between $x^{j,j}(t_k)$ and $x^{j,i}(t_k)$ which are the initial condition of both $P^j(t_k)$ and $P^i(t_k)$, and this will lead to mismatch between the trajectories that $P^j(t_k)$ and $P^i(t_k)$ generate for j-th vehicle. Then, decreasing of the sampling time $\delta^i(t_k)$ which means a faster communication rate will improve the feasibility and stability of the decentralized RHC.

6.5.4 Proposed Algorithm for communication topology optimization

In this section based on the results of the previous section a new improved algorithm for decentralized RHC is proposed. First a sub-algorithm is proposed which allows using the maximum available communication capacity for choosing the best combination of communication topology, communication rate and prediction horizon so that the stability enhancement of decentralized RHC is achieved. Then, this sub-algorithm is merged with the decentralized RHC algorithm. Once for any reason the performance and stability of decentralized RHC architecture is not reasonable, the new communication based approach can be used to enhance the stability. This approach consists of three main steps: 1- the communication topology is changed for enhancement of stability based on the results of section 5.1. 2- Based on the new communication topology the graph topology (section 3) are updated. 3- The communication rate and prediction horizon are selected for the new problem so that the stability is guaranteed (section 5.2 & 5.3). The new sub-algorithm is proposed in figure 7:

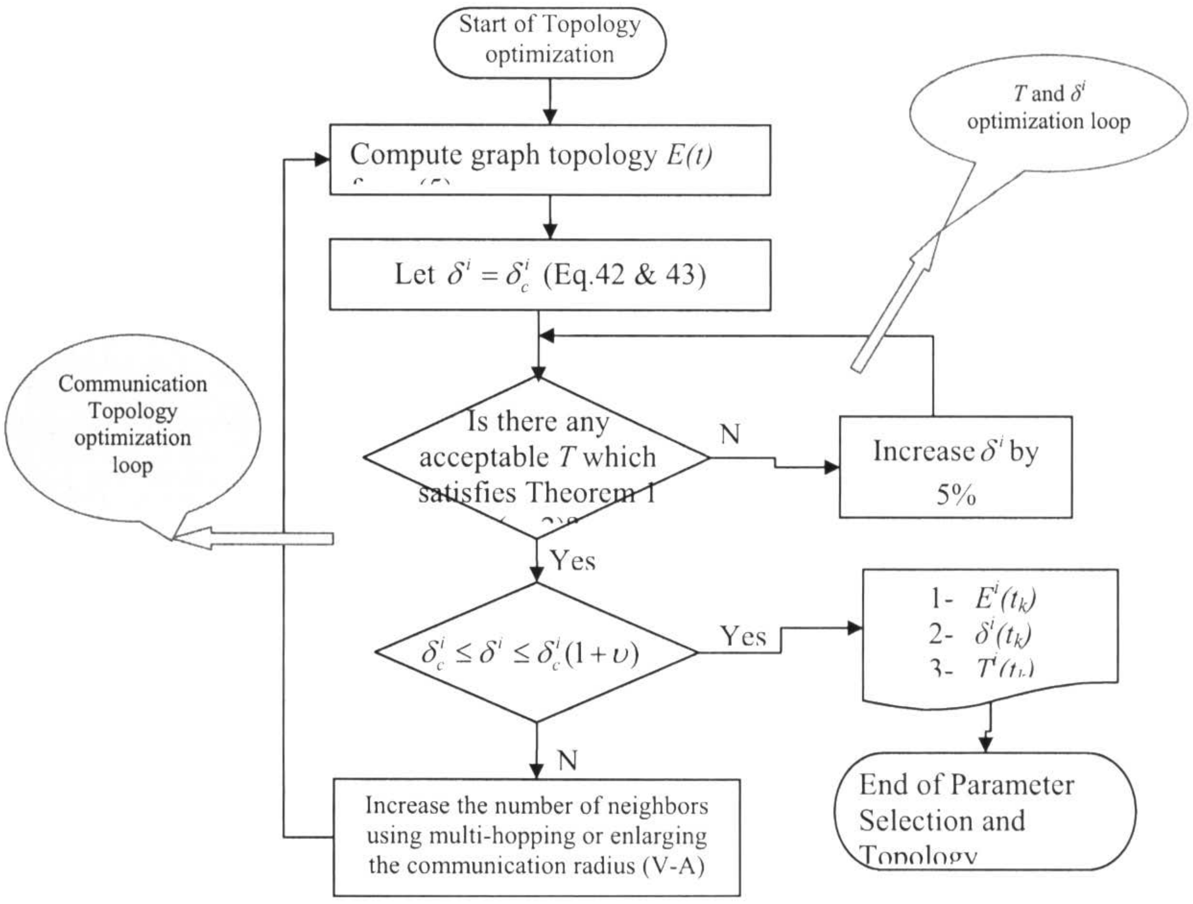

Start of Topology optimization
Compute graph topology $E(t)$
Let $\delta^i = \delta^i_c$ (Eq.42 & 43)
Is there any acceptable T which satisfies Theorem 1
N
Increase δ^i by 5%
T and δ^i optimization loop
Communication Topology optimization loop
Yes
$\delta^i_c \le \delta^i \le \delta^i_c(1+\upsilon)$
Yes
1- $E^i(t_k)$
2- $\delta^i(t_k)$
3- $T^i(t_k)$
N
Increase the number of neighbors using multi-hopping or enlarging the communication radius (V-A)
End of Parameter Selection and Topology

Fig. 7: Sub-algorithm for optimization of graph topology and decentralized RHC architecture

Figure 7 shows an algorithm which optimizes the graph topology and decentralized RHC parameters in order to use maximum communication capacity for improving the stability and performance. More specifically the output of this algorithm is the optimum communication topology, communication rate (execution time) and prediction horizon. In this diagram υ is a positive small number; a bigger υ leads to a more conservative design which doesn't use maximum available communication topology. At each sampling time each vehicle uses this algorithm to find the best communication topology, execution horizon (sampling time) and prediction horizon. It is important to mention that this selection is restricted to available communication capability and the decentralized RHC stability requirements. We call this sub-algorithm as "Communication Topology Optimization Algorithm" (CTOA); we combine the CTOA with the decentralized RHC algorithm as follows:

Modified Decentralized RHC Algorithm:

8) Let $k=0$.

9) **Compute optimized graph topology $E^i(t_k)$, communication rate and prediction horizon by "Communication Topology Optimization Algorithm" (CTOA).**

10) Sample $x^i_{actual}(t_k)$.

11) Communicate $x^i_{actual}(t_k)$ to neighbour vehicles and receive $x^j_{actual}(t_k - d^{ij}\delta^i(t_k))$ where $(i,j) \in E^i(t_k)$.

12) Solve the decentralized optimization problem $P_i(t_k)$ *and* generate the control action for time interval $[t_k, t_k+T(t_k)]$.

13) Execute the control action for individual vehicle over the time interval $[t_k, t_{k+1}]$.

$$u^i(t) = u^{*i}_{t_k}(t);\ t \in [t_k, t_{k+1}] \qquad \textbf{(61)}$$

14) $k=k+1$. Goto step 2.

This modified decentralized RHC algorithm allows using the maximum available communication topology for stability and performance enhancement.

6.6 Example

6.6.1 Case Study I: Formation of a group of mobile robots

The simulation example involves the formation stabilization of a group of five differentially driven wheeled mobile robots, where it is desirable that each robot takes a predefined position and finally reaches an assigned target. Using feedback linearization the equations of a mobile robot can be transformed into a two dimensional double integrator [26,27] as follows:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= u_1 \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= u_2 \end{aligned} \tag{62}$$

where, (x_1, x_3) denotes the position vector, (x_2, x_4) is the set of velocity components, u_1 and u_2 are the inputs to the robot. These dynamics can also serve as a description for a rotorcraft-like UAV flying at constant altitude. Random noise is added to the dynamics to investigate the effect of uncertainty. The following undirected graph topology is used to study the formation stabilization of a group of five robots:

$$\begin{aligned} V &= \{1, 2, 3, 4, 5\} \\ E &= \{(1,2), (2,3), (3,4), (3,5)\} \end{aligned} \tag{63}$$

All matrix penalties in the cost function are taken as the identity matrix except the components corresponding to the velocities are multiplied by 0.001. B-spline basis functions are used to generate the desired trajectories. Furthermore, the concept of flat outputs [8] is utilized to reduce the dimension of the optimization problem and alleviate the online computation burden. The reader is referred to [11] for a comprehensive review on the trajectory generation of the optimization problems.

A "task allocator unit" is also developed that minimizes an overall cost to go for all vehicles and assigns one target to each vehicle. In fact, all the possible cases (permutations) of vehicles to targets are considered and a globally optimal permutation is chosen.

The performance is measured by calculating the final cost of entire group; obviously, it is desired to minimize the final cost for better performance. The optimization toolbox of Matlab is used for solving the optimization problem. Form stability condition (40), T_{comu} for 5 robots is approximated correspondingly as follows 2.5, 2.5, 3.5, 2.5, 2.5 that implies for any prediction horizon smaller than these values the decentralized RHC problem must be stable. To verify that, simulation results for pair $(T,\delta)=(1,0.2)$ (where $T < T_{comu}$) (all time units are sec.) is depicted in figure 8, this figure shows the formation of a group of 5 robots aiming to visit their assigned targets marked by 5 stars on the right part of figure, the initial position of vehicles is shown by 5 circles on figure 8. As it is seen the robots reach their final position precisely; however, figure 9 shows that if we increase the prediction horizon and reduce the communication rate (or increase the execution time) to $(T,\delta)=(5,0.3)$ (where $T > T_{comu}$), the robots don't reach their assigned target; hence, the feasibility is not achieved that verifies the results of theorem 2. However, the stability can be improved in this case by aiding the results of section IV.B (or IV-C) through changing the communication topology; in fact, the communication radius is enlarged to allow more neighbour for each robot, the new graph topology is presented as follows:

$$E=\{(1,2),(2,3),(2,4),(3,4),(3,5),(4,5)\} \quad \textbf{(64)}$$

Figure 10 shows the simulation results for this enhanced graph topology. As is it seen the performance of the entire group is improved by this new graph topology.

The codes related to the simulation of this case are stored in the folder "Decentralized_task10" and the results can be compared with the centralized case stored in folder "Centralized_task_10".

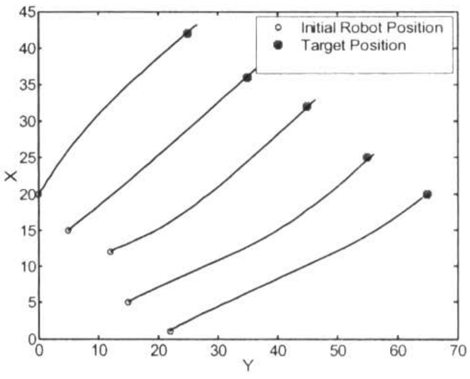


Fig. 8: Formation of a group of robots with $(T,\delta)=(1,0.2)$

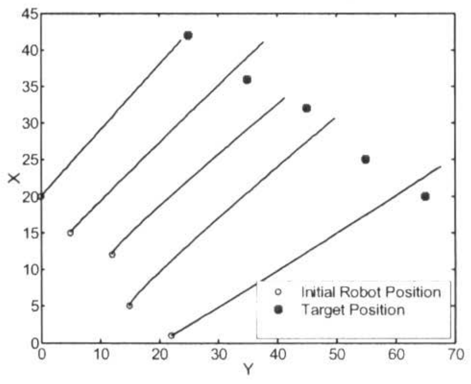


Fig. 9: Formation of a group of robots with bigger prediction horizon and execution time: $(T,\delta)=(5,0.3)$

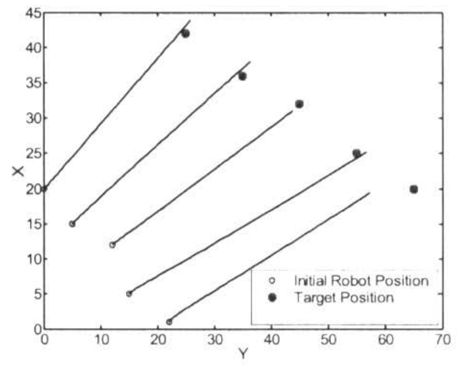


Fig. 10: Formation of a group of robots with larger communication radius and bigger prediction horizon and execution time: $(T,\delta)=(5,0.3)$

6.6.2 Case Study II: Formation of a Fleet of UAVs

In this case a flight formation of a fleet of rotorcraft UAVs with the following 3DOF dynamics is considered [6]:

$$\begin{cases} \dot{u} = vr + c_x u\sqrt{u^2+v^2} - \dfrac{k_{mr_x}}{m} T_{mr} \\ \dot{v} = -ur + c_y v\sqrt{u^2+v^2} \\ \dot{r} = \dfrac{-l_{tr}}{I_{zz}} T_{tr} \\ \dot{\psi} = r \\ \dot{x}_c = u\cos(\psi) - v\sin(\psi) \\ \dot{y}_c = u\sin(\psi) + v\cos(\psi) \end{cases} \tag{65}$$

Where:

$$\begin{cases} c_x = -\dfrac{1}{m}(0.5)\rho_a S_{fus_x} \\ c_y = -\dfrac{1}{m}(0.5)\rho_a S_{fus_y} \end{cases} \tag{66}$$

Where:

$$\begin{cases} S_{fus_x} = 0.1\ m^2 \\ \rho_a = 1.0\, kg/m^3 \\ m = 8.2\, kg \\ k_{mr_x} = -0.3 \\ l_{tr} = 0.91 \\ I_{zz} = 0.28\, kg \times m^2 \end{cases} \tag{67}$$

It is desired to investigate a case when the vehicles leave the formation due to obstacles (or far targets) (see figure 11), and consequently they need a variable communication topology. In fact, the following topologies will be used during performing the mission:

$$\begin{aligned} V &= \{1,2,3\} \\ E(t_0) &= \{(1,2),(1,3),(2,3)\} \\ E(t_1) &= \{(2,3)\} \\ E(t_f) &= \{(1,2),(1,3),(2,3)\} \end{aligned} \qquad \textbf{(68)}$$

This variable graph topology shows that UAVs first fly in formation ($E(t_0)$); then they leave the formation to avoid obstacle ($E(t_1)$) and finally they again return to the formation ($E(t_f)$). This graph topology allows studying a time-varying communication topology.

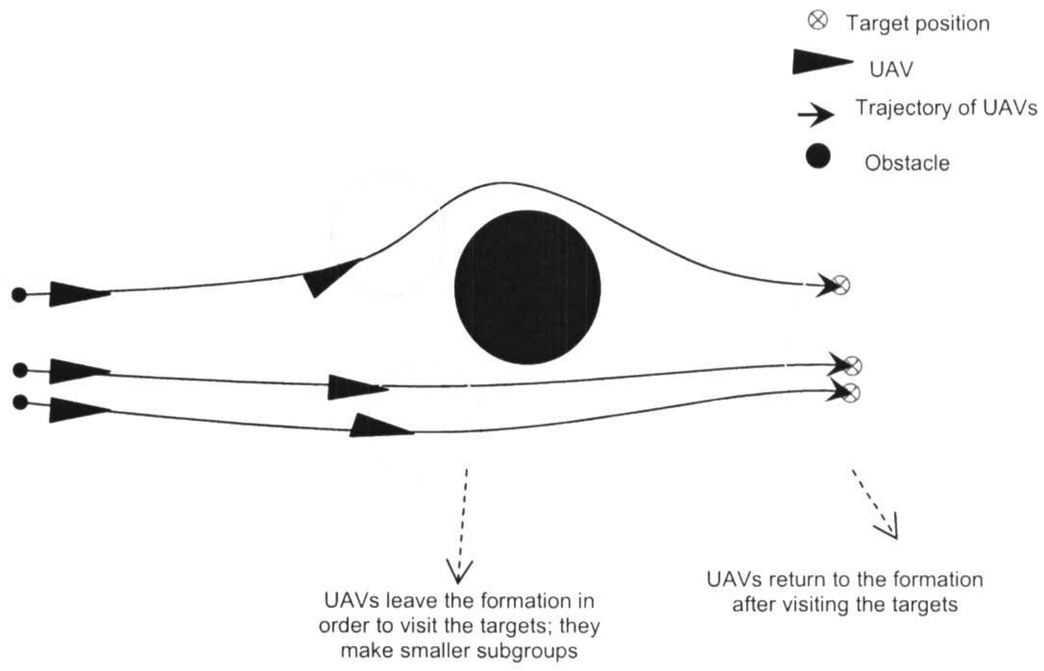

⊗ Target position
UAV
Trajectory of UAVs
● Obstacle
UAVs leave the formation in order to visit the targets; they make smaller subgroups
UAVs return to the formation after visiting the targets

Fig. 11: Formation of a fleet of UAVs: a time varying communication topology due to obstacle

The parameters of the problem are listed below:

$$\begin{cases} T = 1.0 \quad \text{sec.} \\ \delta = 0.05 \quad \text{sec.} \\ Rc = 6 \quad m \\ Rs = 2 \quad m \\ 0 \le T_{mr} \le 1000 \\ 0 \le T_{tr} \le 20 \\ V_{\max} = 10 \; m/\text{sec (Speed Constraint)} \\ Q = diag(0.1, 0.1, 0.1, 0.1, 20, 20) \\ R = diag(0.02, 0.02) \\ P = diag(1, 1, 1, 1, 1, 1) \end{cases} \qquad \textbf{(69)}$$

These values are used for all the vehicles. The forward velocity u and yaw angle ψ will be used as the flat outputs. These flat outputs are parameterized as a function of time by the means of cubic Spline functions [30]. RHCOOL library along with SNOPT optimization package [31] will be used to solve this problem. The obstacle is modeled as a circle and the vehicles are not allowed to collide that. Mathematically, this restriction is implemented by the following constraint:

$$x_c^2(t) + y_c^2(t) > R_{Obs.}^2 = 16 \qquad \textbf{(70)}$$

The trajectory of all vehicles must respect this constraint during performing the mission. The SNOPT optimization package allows non-convex optimization problem with nonlinear constraints; hence applying the above nonlinear constraint is possible by SNOPT. Figure (12) shows the trajectories of this scenario:

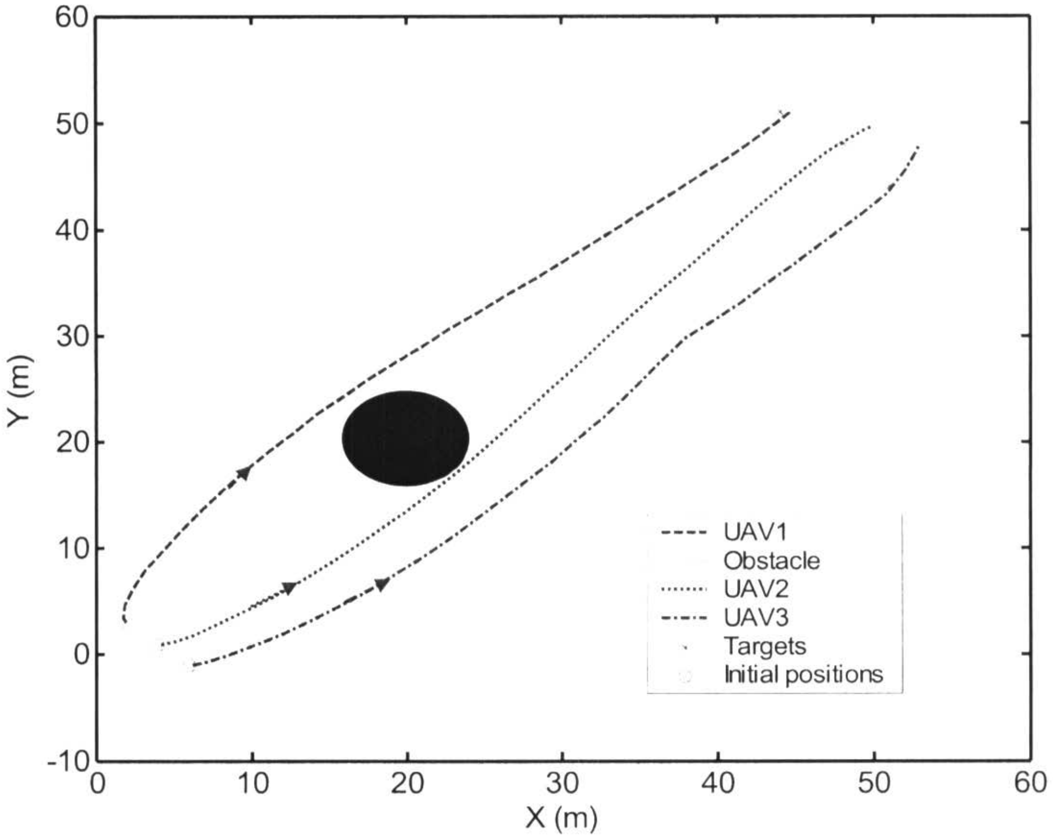


Fig. 12: flight formation of a fleet of UAVs: a time varying communication topology due to obstacle

As it is seen from the figure (12) the UAVs visit their assigned target and avoid the obstacle (non-fly zone) and collision. Figure (13) shows the distance of the vehicle at each time step:

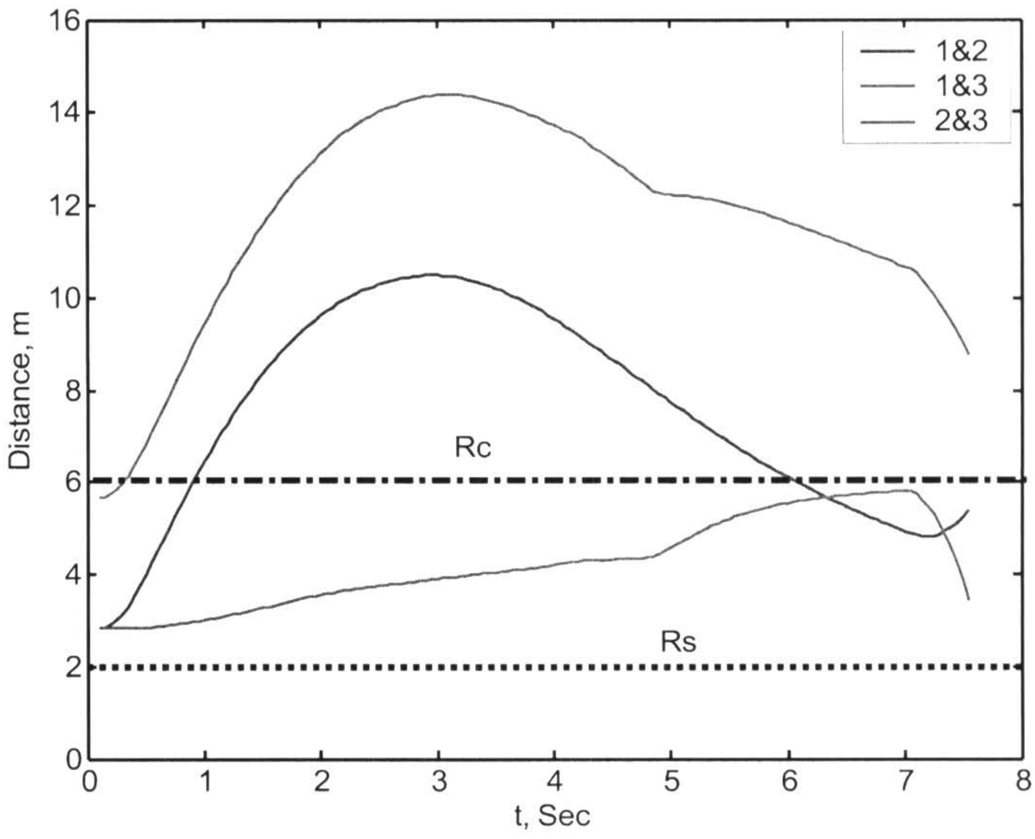


Fig. 13: The distance between vehicles during performing the mission

As it is seen from the above figure the distance between UAV1 and the other two UAVs exceed the communication radius; hence, for some time interval the UAV1 is not the neighbour of the other two UAVs and change the graph topology. Also, as it is seen the UAVs keep the safe distance (*Rs*) and don't contact each other.

The codes related to the simulation of this case are stored in the folder "RHCOOL_Decentralized_task10".

6.7 Conclusion

The effect of variable communication on stability of decentralized RHC of multi-vehicle systems has been investigated. Since each vehicle uses a model of neighbours to predict their trajectory, the effect of model uncertainty on the stability of the entire group is investigated and bounds are found on the *prediction horizon* and *communication rate* that guarantee the stability. The analysis shows that faster communication rates and smaller prediction horizons can improve the stability of decentralized RHC. Furthermore, it is seen that expanding the communication radius and using the multi-hopping communication can improve the stability of decentralized RHC. A modified decentralized RHC algorithm is proposed which allows using the maximum available communication capacity for stability enhancement of decentralized RHC. Two different case studies have been investigated to clarify the problem statement. Future work includes solving some sophisticated case studies with varying communication topology and varying communication rates to verify the theoretical results of this report.

6.8 References

[1]Keviczky T., Borrelli F. and Balas G. J., "Stability Analysis of Decentralized Receding Horizon Control for the decoupled systems ", *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, Seville, Spain, December 12-15, 2005, pp. 1689-1694.

[2]Dunbar W. B. and Murray R. M., "Receding horizon control of multi-vehicle formations: A distributed implementation", Proceedings *of 43rd IEEE Conf. on Decision and Control*, 2004.

[3]Shih-Ho W. and Davison, E. "On the stabilization of decentralized control systems", *IEEE Transactions on Automatic Control*, Vol. 18, Issue 5, Oct 1973, pp. 473 – 478.

[4]Shi L., Ko C. K., Jin Z., Gayme, Gupta D., Waydo V. and Murray, R.M., "Decentralized control across bit-limited communication channels: an example",

Proceedings of the American Control Conference, 8-10 June 2005, vol. 5, pp. 3348 - 3353.

[5]Shim D. H., Kim H. J. and Sastry S., "Decentralized Reflective Model Predictive Control of Multiple Flying Robots in Dynamic Environment", Department of Electrical Engineering & Computer Sciences University of California at Berkeley, Tech report 2003.

[6]Richards A. and How J.,"A decentralized algorithm for robust constrained model predictive control", *Proceedings of American Control Conference*, 2004.

[7]De Gennaro, M.C and Jadbabaie, A., "Formation Control for a Cooperative Multi-Agent System using Decentralized Navigation Functions", *American Control Conference,* 2006, June 14-16, 2006, pp. 1346 – 1351.

[8]Franz R., Milam M. and Hauser J., "Applied Receding Horizon Control of the Caltech Ducted Fan," *IEEE ACC*, Anchorage, 2002.

[9]Mayne D. Q., Rawlings J. B., Rao C. V. and Scokaert P. O. M., "Constrained model predictive control: Stability and optimality", *Automatica*, 36, 2000, pp. 789-814.

[10]Kuwata Y., Richards A., Schouwenaars T., and How J. "Decentralized Robust Receding Horizon Control for Multi-vehicle Guidance", *Proceedings of the 2006 American Control Conference,* Minneapolis, Minnesota, USA, June 14-16, 2006.

[11]Kuwata Y., Schouwenaars T., Richards A., and How J, "Robust Constrained Receding Horizon Control for Trajectory Planning" in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2005.

[12]Richards A. and How J. P., "Implementation of Robust Decentralized Model Predictive Control", *AIAA Guidance, Navigation and Control Conference,* 2005.

[13] H. Izadi, B. W. Gordon, C. A. Rabbath, "A Time-Varying Formulation and Stability Analysis for Decentralized Receding Horizon Control of Multi-Vehicle Systems", Submitted to the 46th IEEE Conference on Decision and Control (CDC), New Orleans, LA, USA, December 12-14, 2007.

[14] H. Izadi, B. W. Gordon, C. A. Rabbath, "Decentralized Receding Horizon Control of Multi-Vehicle Systems Using a Variable Communication Algorithm", Submitted to the 46th IEEE Conference on Decision and Control (CDC), New Orleans, LA, USA, December 12-14, 2007.

[15] H. Izadi, B. W. Gordon, C. A. Rabbath, "Stability Improvement for Time Varying Decentralized Receding Horizon Control Systems", Submitted to 13th IEEE IFAC International Conference on Methods and Models in Automation and Robotics (MMAR), Szczecin, Poland, 27 - 30 August 2007.

[16] H. Izadi, B. W. Gordon, C. A. Rabbath, "A Variable Communication Approach for Decentralized Receding Horizon Control of Multi-Vehicle Systems", Accepted to be presented in 2007 American Control Conference (ACC07), New York City, USA, July 11-13, 2007.

[17]Borrelli F., Keviczky T., Fregene K. and Balas G., "Decentralized Receding Horizon Control of Cooperative Vehicle Formations", *ECC and 44th IEEE Conference on Decision and Control*, 12-15 Dec. 2005, pp. 3955 – 3960.

[18]Hill P. E., Murray W. and Wright M. H., Practical optimization, Academic Press, page 65-66, 1981.

[19]Jadbabaie A. and Hauser J., "On the Stability of Receding Horizon Control with a General Terminal Cost", *IEEE Transactions on Automatic Control*, Vol. 50, No. 5, May 2005, pp. 674-678.

[20]Dunbar W., "Distributed Receding Horizon Control of Multiagent Systems", PhD thesis, California Institute of Technology, April, 2004.

[21] Kågström B., "Bounds and perturbation bounds for the matrix exponential", BIT Numerical Mathematics Journal, Vol. 17, N0. 1, March, 1997, pp. 39-57.

[22]Kramer G., Gupta P. and Gastpar M., "Information-theoretic Multi-hopping for Relay Networks", *Int. Zurich Seminar on Communications (IZS)*, Feb. 18-20, 2004.

[23]Zhang F., Grocholsky B., Kumar R. and M. Mintz, "Cooperative Control for Localization of Mobile Sensor Networks" GRASP Laboratory, University of Pennsylvania, Internal paper, 2003.

[24]Gouaisbaut F., Dambrine M. and Richard J.P., "Robust control of delay systems: a sliding mode control design via LMI", *Elsevier Systems & Control Letters* 46, (2002), pp. 219 – 230.

[25]Kirk D. E., "Optimal Control Theory: An Introduction", Dover Publications, April 2004.

[26]Young B. J., Beard R. W. and Kelsey J. M., "A control scheme for improving multi-vehicle formation maneuvers", *American Control Conference*, Arlington, VA, pp. 704-709, 2001.

[27]Desai J. P., Ostrowski J. and Kumar V., "Controlling formations of multiple mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998, pp. 2864-2869.

[28]Milam M. B., "Real-time optimal trajectory generation for constrained dynamical systems", *PhD Thesis*, California Institute of Technology, Pasadena, CA, 2003.

[29]Gavrilets V., Mettler B., Feron E. "Dynamic Model for a Miniature Aerobatic Helicopter", MIT Internal Report.

[30]Stephen A. Dyer and Justin S. Dyer, "Cubic-Spline Interpolation: Part 1", *IEEE Instrumentation & Measurement Magazine*, March 2001, pp.44-46.

[31]GILL P. E., MURRAY W. and SAUNDERS M. A., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming", Department of Mathematics University of California, San Diego, La Jolla, CA 92093-0112, USA, May 30, 2006.

CHAPTER 7: REAL-TIME DYNAMIC SCHEDULING ALGORITHM FOR MULTIPLE RHC RUNNING ON DISTRIBUTED DIGITAL PROCESSORS

(Task #11)

Ali Azimi
PhD Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory
Department of Mechanical and Industrial Engineering
Concordia University
Montreal, Quebec, Canada H3G 1M8

March 2007

Abstract

This report documents the work related to tasks #11 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". This report describes the design of a real time scheduling algorithm for computation and communication scheduling of multiple uncertain receding horizon control systems, running on multiple distributed computers. In this work, execution horizon and communication period of all systems are determined such that directly optimize robust RHC performance of the whole network. This is achieved through minimization of the upper bound on the so-called *overall closed-loop cost* of the system, subject to scheduling constraints on both computation and communication. The performance of the method is demonstrated by simulating it in Visual C++ by using RHC Object Oriented Library (RHCOOL), which is adapted for Dynamic Scheduling of Decentralized RHC Systems.

7.1 Introduction

Scheduling of RHC computational and communication is very important for decentralized RHC control since it allows optimization of resources to achieve maximum performance in the presence of uncertainty, disturbances, and delays. The tasks addressed in this report related to the project entitled "Real-time Scheduling of Multiple Uncertain Receding Horizon Control Systems" is as follows:

Task 11: Design a scalable, dynamic run-time scheduling algorithm for multiple RHC running on distributed digital processors.

In this report, we consider the problem of controlling multiple uncertain nonlinear systems by means of concurrent RHC schemes. The systems considered in this report are dynamically decoupled. However, because of being in a network, the coupling term is appeared in the controller cost function. In addition, the proposed formulation is adapted for decentralized application, so the communication delay is also accounted for in this report.

A new scheduling approach is proposed by combining the results from continuous time nonlinear systems theory and the concept of Rate Monotonic Priority Assignment (RM) [3]. Assuming the prediction horizon is a known constant and using a single computing resource, the new technique determines the execution horizons of all RHC systems. The execution horizon determination of each subsystem while optimizing the performance is cast into a constrained optimization problem. The robust performance is formulated in the objective function and the schedulability condition is guarantied using a constraint. Online solution of the foresaid optimization problem, using the updated optimization parameters, results in dynamically determining the execution horizons. This report can be regarded as the first systematic approach for dynamic scheduling of multiple RHC systems subject to computing and communicating resource limitations and model uncertainties.

The report is organized as follows:

Chapter 2 presents the dynamic scheduling of decoupled RHC systems using a single processor. It is supposed that the systems have no data exchanging and they are completely decoupled. In this chapter, after problem formulation and scheduling formulations, a discussion is presented for performance improvement by using a new upper bound instead of Gronwall-Bellman Lemma [4].

In Chapter 3, the systems are considered to be coupled in their controller cost functions. The scheduling algorithm presented in this section is applicable to the coupled systems on a single processor.

Chapter 4 is the general case, the communication delay is also added, and a new method for communication scheduling is presented. The scheduling algorithm proposed in this section is using the results of previous sections and is applicable for scheduling of multiple RHC systems on distributed processors. In this method, computation and communication scheduling is formulated by optimally determining the execution horizon and communication period of all systems in the network. Summary and future works are presented in Chapter 5 and the references are in Chapter 6.

7.2 Scheduling of Multiple RHC Systems without Coupling on a Single Processor

Receding horizon control (RHC) is a repeated online solution of a finite horizon open-loop optimal control problem [**1**]. Based on the currently available state values, the optimal control problem is solved and the resulting control input is applied to the plant in a period called the *execution horizon*. In real-time implementation of a RHC problem, the execution horizon is the closed-loop sampling period. It should be selected carefully to obtain a suitable trade-off between computational expense and controller performance.

Application of RHC to control problems with multiple subsystems such as multiple Unmanned Aerial Vehicle (UAV) systems can be addressed by applying RHC to the individual subsystems. This results in multiple RHC processes that must be scheduled in an appropriate manner to achieve optimal performance in the presence of computing resource limitations. Systematic methods for scheduling RHC systems are typically not discussed in the literature, despite the fact that computational delays resulting from scheduling can dramatically affect stability and performance. A dynamic scheduling approach for multiple discrete-time decoupled linear RHC systems has been previously developed [5]. This method assigns the priorities to different subsystems based on the value of a computational delay dependent cost index. Premature termination of the optimization process is employed. However, in [5], uncertainties in the subsystems are not explicitly accounted for. The authors in [6] considered the problem of optimal on-line assignment of sampling periods for a set of linear-quadratic (LQ) controllers. They used feedback from the plant states to distribute the computing resources optimally among the tasks.

In this chapter, we consider the problem of controlling n decoupled uncertain nonlinear systems by means of concurrent RHC schemes. A new scheduling approach is proposed by combining the results from continuous time nonlinear systems theory and the concept of Rate Monotonic Priority Assignment (RM) [3]. Assuming the prediction horizon is a known constant and using a single computing resource, the new technique determines the execution horizons of all RHC systems. The execution horizon determination of each subsystem while optimizing the performance is cast into a constrained optimization problem. The robust performance is formulated in the objective function and the schedulability condition is guarantied using a constraint. Online solution of the foresaid optimization problem, using the updated optimization parameters, results in dynamically determining the execution horizons.

7.2.1 Problem Statement

7.2.1.1 RHC formulation

Consider the following nominal equation:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), t\right) \tag{1}$$

which serves as a model for the actual system described by:

$$\dot{\hat{\mathbf{x}}} = \mathbf{f}\left(\hat{\mathbf{x}}(t), \mathbf{u}(t), t\right) + \mathbf{g}(\hat{\mathbf{x}}, \mathbf{u}, t) \tag{2}$$

where $\mathbf{x}(t) \in \Re^{n}$ and $\hat{\mathbf{x}}(t) \in \Re^{\hat{n}}$ are the nominal and actual states of the system, respectively. The input vector $\mathbf{u}(t) \in \Re^{m}$ satisfies the constraints $\mathbf{u}(t) \in U$ $(\forall t \geq 0)$, where U is the allowable set of inputs. Furthermore, it is assumed that (A1-A3) in [2] are also satisfied; that is, f is twice differentiable, U is compact and convex, and system (**1**) has a unique solution for a given initial condition. The finite horizon cost (from initial state $\mathbf{x}(t)$) is defined as follows [7]:

$$J_T(\mathbf{x}(t), \mathbf{u}(\cdot)) = \int_t^{t+T} q\left(\mathbf{x}^{\mathbf{u}}\left(\tau; \mathbf{x}(t)\right), \mathbf{u}(\tau)\right) d\tau + V\left(\mathbf{x}^{\mathbf{u}}\left(t+T; \mathbf{x}(t)\right)\right) \tag{3}$$

where T is the optimization horizon of the RHC controller. The optimal cost is given by:

$$J_T^*(\mathbf{x}(t)) = \inf_{\mathbf{u}(\cdot)} J_T(\mathbf{x}(t), \mathbf{u}(\cdot)) \tag{4}$$

The optimized trajectory resulting from (4) is defined as $(\mathbf{x}_T^*(\tau; \mathbf{x}(t)), \mathbf{u}_T^*(\tau; \mathbf{x}(t)))$, $\tau \in (t, t+T]$. In the closed loop RHC the calculated input $\mathbf{u}_T^*(\tau; \mathbf{x}(t))$ is applied to the actual system (2), and $\tau \in (t, t+\delta]$, while δ is called the Execution Horizon ($\delta < T$).

7.2.1.2 Real-time scheduling of multiple RHC systems

Consider the problem of controlling n decoupled uncertain nonlinear systems using the RHC scheme. The following equations serve as the nominal model for describing the decoupled plants

$$\dot{\mathbf{x}}_i = \mathbf{f}_i\left(\mathbf{x}_i(t), \mathbf{u}_i(t), t\right), \quad i = 1, \ldots, n \tag{5}$$

in addition, the actual plants are described by

$$\dot{\hat{\mathbf{x}}}_i = \mathbf{f}_i\left(\hat{\mathbf{x}}_i(t), \mathbf{u}_i(t), t\right) + \mathbf{g}_i(\hat{\mathbf{x}}_i, \mathbf{u}_i, t), \quad i = 1, \ldots, n \tag{6}$$

For RHC control of a single apparatus using a single computer the calculated inputs are applied to the system for a period equal to the execution horizon of the system. Therefore, in real-time implementation, a common way is to define a real-time periodic task [3], with its period equal to the execution horizon of the system.

Suppose that the systems described in (5) and (6) are connected to a single computer for feedback control. From a computer control point of view, each control system can be handled as a periodic task in the real-time programming. The period of each periodic task is equal to the execution horizon of its related subsystem. Determining these periods (or execution horizons) is not trivial. In this report, a systematic approach is presented to calculate these periods.

For the proposed approach, the execution horizons of all subsystems should be defined such that the overall performance of the system is maximized. In order to evaluate the performance of the system, the following cost function is proposed as the cost of the closed loop system from time t to $t+T_2$, where T_2 is the period that calculated execution horizons would be applied to

$$\hat{J}_{T_2}=\sum_{i=1}^{n}\left(\sum_{k=1}^{d_i}\left(\int_{t_k^i}^{t_k^i+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}^*_{T,i}\left(\tau;\hat{\mathbf{x}}_i\left(t_k^i\right)\right)\right)d\tau\right)+\int_{t+d_i\delta_i}^{t+T_2} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}^*_{T,i}\left(\tau;\hat{\mathbf{x}}_i\left(t+d_i\delta_i\right)\right)\right)d\tau\right) \tag{7}$$

where $d_i=\lfloor T_2/\delta_i\rfloor$[32], $t_k^i=t+(k-1)\delta_i$, and $\mathbf{u}^*_{T,i}\left(\tau;\hat{\mathbf{x}}_i\left(t_k^i\right)\right)$ is the optimal input applied to subsystem i.

The idea is to find the execution horizon of each subsystem (δ_i), such that $\sum_{i=1}^{n}\hat{J}_{T_2}^i$ is minimum.

The scheduling algorithm that is presented in this report is based on the concept of Rate Monotonic Priority Assignment (RM) [3]. The system is schedulable using RM, for a set of n tasks, if the following inequality is valid [3]:

$$\mu=\sum_{i=1}^{n}\frac{\delta_{c,i}}{p_i}\leq n(2^{\frac{1}{n}}-1) \tag{8}$$

where μ is CPU Utilization factor, $\delta_{c,i}$ is the computation time of subsystem i, and p_i is the period of task i, which is equal to the execution horizon of subsystem i ($p_i=\delta_i$).

Remark 1: In this report, we propose a dynamic scheduling algorithm, which updates the execution horizons for the period of time $T_2>\delta_i$; so we force the proposed scheduling method to be valid under the condition of (8) and it guarantees schedulability of the system.

[32] Floor function of a real number x, denoted $\lfloor x\rfloor$, is the largest integer less than or equal to x.

7.2.2 Model Based Dynamic Scheduling

Equation (7) needs the future states of the actual systems and the future optimized inputs (from time t to $t+T_2$) for calculating $\hat{J}_{T_2}$ which are not available at current time (t). Therefore, instead of calculating $\hat{J}_{T_2}$, the following cost is proposed which is an estimation of the cost in (7):

$$\bar{J}_{T_2}(\hat{\mathbf{x}}(t),\mathbf{u}_T^*(.)) = \sum_{i=1}^{n}\left(\frac{T_2}{\delta_i}\int_t^{t+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}_{T,i}^*(\tau;\hat{\mathbf{x}}_i(t))\right)d\tau\right) \tag{9}$$

Proposition1:

Suppose the following assumptions hold true:

1. q_i is quadratic, so: $q_i(\mathbf{x}_i,\mathbf{u}_i) = \mathbf{x}_i^T Q_i \mathbf{x}_i + \mathbf{u}_i^T R_i \mathbf{u}_i$
2. $\overline{Q}_i(\mathbf{x}_i) = \mathbf{x}_i^T Q_i \mathbf{x}_i$ is Lipschitz continuous with constant P_i.
3. $\mathbf{f}_i$ in equation (5) is Lipschitz continuous with constant L_i.
4. $\mathbf{g}_i$ in equation (6) is bounded and $\|\mathbf{g}_i(\mathbf{x}_i,\mathbf{u}_i)\| \le b_i$.

Then the following inequality exists:

$$\begin{aligned}&\int_t^{t+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right)d\tau \le \\ &\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*(\tau;\mathbf{x}_i(t)),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right)d\tau + \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right)\end{aligned} \tag{10}$$

where $b_{s,i}$ is the bound on measurement errors caused by sensor noise.

Proof:

From assumption 1:

$$\int_t^{t+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right)d\tau = \int_t^{t+\delta_i}\overline{Q}_i(\hat{\mathbf{x}}_i(\tau))d\tau + \int^{t+\delta_i}\left(\mathbf{u}_{T,i}^*(.)\right)^T R_i\mathbf{u}_{T,i}^*(.)d\tau \tag{11}$$

From Lipschitz continuity of $\overline{Q}_i$:

$$A = \int_t^{t+\delta_i}\left(\overline{Q}_i(\hat{\mathbf{x}}_i(\tau))-\overline{Q}_i(\mathbf{x}_i(\tau))\right)d\tau \le \int_t^{t+\delta_i} P_i\|\hat{\mathbf{x}}_i(\tau)-\mathbf{x}_i(\tau)\|d\tau \tag{12}$$

Using the Gronwall-Bellman inequality [4]:

$$A \le P_i \int_t^{t+\delta_i} \left(e^{L_i(\tau-t)} \left\| \hat{\mathbf{x}}_i(t) - \mathbf{x}_i(t) \right\| + \frac{b_i}{L_i}\left(e^{L_i(\tau-t)} - 1\right) \right) d\tau \tag{13}$$

The term $\left\| \hat{\mathbf{x}}_i(t) - \mathbf{x}_i(t) \right\|$ denotes the error in state measurement. It is assumed that the measurement noise is the only error source with the bound $b_{s,i}$ such that

$$\left\| \hat{\mathbf{x}}_i(t) - \mathbf{x}_i(t) \right\| \le b_{s,i} \tag{14}$$

Therefore, from (13) and (14):

$$A \le P_i \int_t^{t+\delta_i} \left(e^{L_i(\tau-t)} b_{s,i} + \frac{b_i}{L_i}\left(e^{L_i(\tau-t)} - 1\right) \right) d\tau = \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i} - 1\right) + b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i} - 1\right) - \delta_i \right)\right) \tag{15}$$

So, from (11), (12), and (15):

$$\begin{aligned} &\int_t^{t+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau), \mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right) d\tau \\ &\le \int_t^{t+\delta_i} \overline{Q}_i\left(\mathbf{x}_i(\tau;\mathbf{x}_i(t))\right) d\tau + \int_t^{t+\delta_i} \left(\mathbf{u}^*_{T,i}(.)\right)^T R_i \mathbf{u}^*_{T,i}(.) d\tau + \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i} - 1\right) + b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i} - 1\right) - \delta_i \right)\right) \end{aligned} \tag{16}$$

In addition, $\mathbf{u}^*_{T,i}(.) = \mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))$ so:

$$\int_t^{t+\delta_i} \overline{Q}_i\left(\mathbf{x}_i(\tau;\mathbf{x}_i(t))\right) d\tau + \int_t^{t+\delta_i} \left(\mathbf{u}^*_{T,i}(.)\right)^T R_i \mathbf{u}^*_{T,i}(.) d\tau = \int_t^{t+\delta_i} q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)), \mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right) d\tau \tag{17}$$

which is a part of optimal open loop cost, calculated at time t. Combination of (16) and (17) result in (10). □

Proposition 2:

Consider *n* decoupled systems with equations presented in (5) and (6), controlled by RHC using only one computer. In addition, the assumptions of proposition 1 are valid. Since uncertainties in the subsystems are different and the measurements are performed with bounded sensor noise, show that the following equation can be used to optimally determine the execution horizons of all subsystems:

$$\min_{\delta_i} \sum_{i=1}^{n} \frac{\alpha_i}{\delta_i} \left(\begin{array}{l} \int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right),\mathbf{u}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right)\right)d\tau \\ +\frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right) \end{array} \right) \tag{18}$$
$$\text{s.t.} \begin{cases} \sum \frac{\delta_{c,i}}{\delta_i} \le n(2^{\frac{1}{n}}-1) \\ \delta_{lb,i} < \delta_i \le \delta_{ub,i} \end{cases}$$

where $\delta_{ub,i}$ and $\delta_{ub,i}$ are the maximum and minimum acceptable execution horizons for subsystem *i*, respectively. $\delta_{p,i}$ is the execution horizon of subsystem *i* before being updated.

Proof:

From (9) and result of proposition 1, the following inequality is valid:

$$\bar{J}_{T_2}(\hat{\mathbf{x}}(t),\mathbf{u}_T^*(.)) \le$$
$$\sum_{i=1}^{n} \frac{T_2}{\delta_i}\left(\int_t^{t+\delta_i} q_i\left(\mathbf{x}_i(\tau;\mathbf{x}_i(t)),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right)d\tau + \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right)\right) \tag{19}$$

where as explained in (9) the left hand side is the estimation of overall cost function presented in (7). Therefore, by minimizing the left hand side, the performance of the system improves. In addition, as explained in Remark 1, equation (8) must hold to guarantee schedulability of the system using RM method. Therefore, it is a straightforward to propose the following for scheduling algorithm, considering α_i as weights applied in each subsystem:

$$\min_{\delta_i} \sum_{i=1}^{n} \frac{\alpha_i}{\delta_i}\left(\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*(\tau;\mathbf{x}_i(t)),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right)d\tau + \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right)\right)$$
$$\text{s.t.} \begin{cases} \sum \frac{\delta_{c,i}}{\delta_i} \le n(2^{\frac{1}{n}}-1) \\ \delta_{lb,i} < \delta_i \le \delta_{ub,i} \end{cases} \tag{20}$$

However, the term $\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*(\tau;\mathbf{x}_i(t)),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right)d\tau$ cannot be calculated directly, since it needs the optimal inputs and states for time *t*, but the execution horizons must be defined before time *t*. So, the following term is proposed instead:

$$\int_t^{t+\delta_i} q_i\left(\mathbf{x}^*_{T,i}\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right),\mathbf{u}^*_{T,i}\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right)\right)d\tau \tag{21}$$

that uses the optimal states and inputs from previous open-loop cost calculation. Combination of (20) and (21) completes the proof. ☐

7.2.3 Dynamic Scheduling Algorithm

In the previous section, a preliminary cost function was presented in (18) which can predict the execution horizons of all subsystems, based on the limited available computational resource. In this section, the updated form of cost function is presented which considers the delay because of computation time. Furthermore, an algorithm is presented to update the scheduling parameters.

Before proposing the algorithm, the delay caused by computation should be handled. Based on the method proposed in [9], for controlling one system by the RHC method, the computation is performed in the period of time t to $t+\delta$, and the result is applied to the period of $t+\delta$ to $t+2\delta$. For the dynamic scheduling of multiple systems, we propose the following method by considering the computational time of each subsystem. Consider Figure **1**, which illustrates the method for two subsystems.

Let us consider time t as the time that the scheduler starts again. As shown in this figure, t_1^p is the time that first subsystem was started its last computation and is the same as the time that the last sampled data of subsystem 1 became available. Consequently, t_2^p is the similar time for subsystem 2. The computation of scheduler starts right after completing computation of both subsystems. Therefore, the scheduler can be considered as a periodic function with a fixed period T_2 and the lowest priority comparing to the periodic functions of all subsystems.

Remark 2. T_2 must be selected such that the execution horizons of all subsystems remain less than T_2, i.e. $\delta_i < T_2$; $1 \le i \le n$. In addition, the closed-loop cost in the interval of t to $t+T_2$ (equation (7)) has been estimated linearly with the interval t to $t+\delta_i$ in equation (9). Therefore, T_2 should be selected such that not being far larger than δ_i.

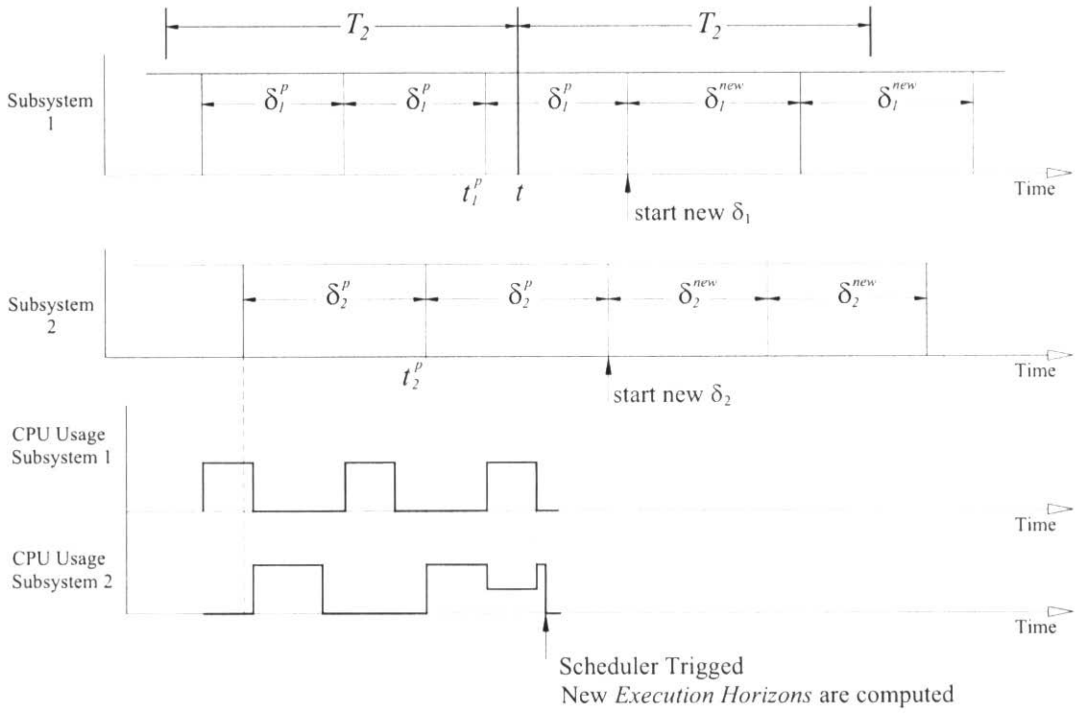


Figure 1- Schematic diagram illustrating the dynamic scheduling procedure

If the computation of Scheduler optimization problem is considerable, the execution horizon of the last computed subsystem (before trigging scheduler) can be increased by $t_{c,s}$ while $t_{c,s}$ is the upper bound in computational time needed to solve scheduler optimization problem.

Based on the foresaid explanations the modified scheduling optimization problem can be expressed as follows:

$$\begin{aligned}
&\min_{\delta_i} \sum_{i=1}^{n} \frac{\alpha_i}{\delta_i} \left(\begin{aligned} &\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t_i^p\right)\right), \mathbf{u}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t_i^p\right)\right)\right)d\tau \\ &+\frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right) \end{aligned} \right) \\
&\text{s.t.} \begin{cases} \sum \dfrac{\delta_{c,i}}{\delta_i} \le n(2^{\frac{1}{n}}-1) \\ \delta_{lb,i} < \delta_i \le \delta_{ub,i} \end{cases}
\end{aligned} \tag{22}$$

Remark 3: This algorithm must be updated for every T_2 while the states must remain in a specified region (box) $\left|\mathbf{x}_i - \mathbf{x}_{c,i}\right| \le \mathbf{R}_i$, where $\mathbf{x}_{c,i}$ defines the center and $\mathbf{R}_i$ presents the dimensions of the box. It must be noted that the scheduling parameters, P_i, b_i, and L_i are functions of states, so $P_i = P_i(\mathbf{x}_{c,i})$, $b_i = b_i(\mathbf{x}_{c,i})$, and $L_i = L_i(\mathbf{x}_{c,i})$ for the predefined $\mathbf{R}_i$.

The procedure for dynamic scheduling can be expressed as follows:

1. Based on the current states and predefined $\mathbf{R}_i$, create a domain, in which the states will remain for the subsequent period T_2.
2. Calculate $P_i = P_i(\mathbf{x}_{c,i})$, $b_i = b_i(\mathbf{x}_{c,i})$ and $L_i = L_i(\mathbf{x}_{c,i})$. See section (IV-B)
3. Estimate $\delta_{c,i}$ from previous computational time. See section (IV-A)
4. Solve equation (22) and calculate δ_i for all subsystems.
5. Update the next execution horizons by the calculated δ_i from step 4.
6. Wait for the next T_2 seconds and repeat the procedure from step 1.

Computation time estimation

Different methods can be used to estimate less conservative upper bound for computation time. Since, this issue is not the focus of this report, a simple method is used which uses the previous $\delta_{c,i}$ to estimate the new one. Based on this method, the maximum $\delta_{c,i}$ from time $t - T_3$ to t is considered as the basis of upper bound calculation for $\delta_{c,i}$ from time t to $t + T_2$. T_3 should be defined in the design process.

7.2.4 Discussion on Performance

In this section, the proposed dynamic scheduling algorithm is briefly reviewed and the validity and performance of the method is discussed. This method was originated from the following three main steps:

1. An execution horizon dependent cost function was proposed in (7). This cost function was estimated by another cost function and estimated in (9). The last cost was used as the main objective for minimization in the rest of the algorithm.
2. The upper bound of that cost index was found in Proposition 1.
3. For the dynamic scheduling algorithm, we minimized the upper bound and claimed that this minimization results in the minimization of the cost function presented in (9).

However, the claim made in the final step is valid if the proposed bound (step 2) is close enough to the actual value of (9). In other form, if the proposed upper bound is far away from the actual value of (9), then minimization of the bound does not affect the actual value of (9), i.e., it does not affect the performance.

In the proof of Proposition 1, the term $\|\hat{\mathbf{x}}_i(t+\delta_i) - \mathbf{x}_i(t+\delta_i)\|$ which is the state estimation error, was replaced by its upper bound presented in [4], which is a result of the Gronwall-

Bellman (GB) Lemma. For a specific problem with a distinct nonlinear nominal function and bounded unmodeled dynamics, finding a less conservative bound is possible. In this section, the foresaid bound is calculated for some systems and compared to the bound results from GB Lemma.

Later on, an algorithm is developed to estimate the upper bound as a function of δ_i, offline. Afterwards, in online programming, the parameters of f_b is updated based on the response of the system, and used for dynamic scheduling.

Before digging into the examples, the problem formulation is presented.

7.2.4.1 Problem formulation

Suppose that an identified model of the physical (actual) system is available. This model is called *actual model* (refer to equation (2)). This model can have bounded uncertainties in parameters. In addition, a simplified control oriented model, which we call it *nominal model* is also available (refer to equation (**1**)). The goal is to find f_b as a function of δ (i.e., $f_b(\delta)$) such that

$$\left\|\hat{\mathbf{x}}(t+\delta)-\mathbf{x}(t+\delta)\right\| \leq f_b(\delta)$$

where $\mathbf{x}(t)$ is available by measurement. Two cases are considered in this study:

1- If there is no error in measurement, then $\left\|\hat{\mathbf{x}}(t)-\mathbf{x}(t)\right\|=0$

2- If bounded noise exists in measurement, then $\left\|\hat{\mathbf{x}}(t)-\mathbf{x}(t)\right\| \leq b_s(\hat{\mathbf{x}})$, where b_s is the bound on sensor noise and it can be dependent on the state of the system in general.

Note that f_b is a function of δ and states (i.e., $f_b(\delta, \mathbf{x})$). First we suppose that f_b is only a function of δ and we find $f_b(\delta)$ such that it is valid for the entire acceptable state space. After that, the parameters of that function are updated in online programming, using the measurement data. Therefore, finding f_b casts into two problems:

Problem 1:

Finding the form of bound function ($f_b(\delta)$), i.e., to determine that if $f_b(\delta)$ is an exponential function or it is a second order polynomial or any other forms. This task is done offline. $f_b(\delta)$ can be found by

- numerical simulations
- analytically by solving the math problem, for every special system

In this report, the Van der Pol oscillator is simulated as an example and a polynomial is proposed instead of Gronwall-Bellman Lemma (GB).

Problem 2:
Updating the parameters of $f_b(\delta)$, online. For example, if $f_b(\delta)$ is the same as the bound presented by GB, then b and L should be found online and used in dynamic scheduling.

One way to find (update) the parameters of $f_b(\delta)$, online, is using the previous data for estimation. $f_b(\delta)$ is a function of δ or *time* and it is an upper bound on $\|\hat{\mathbf{x}}(\delta)-\mathbf{x}(\delta)\|$. Therefore, for estimating the parameters of $f_b(\delta)$, the term $\|\hat{\mathbf{x}}(\delta)-\mathbf{x}(\delta)\|$ should be known for different values of δ.

In this report, the example is solved by having an offline lookup table. In the examples that will be presented in the next report, the online estimation of parameters will be used.

7.2.4.2 Van Der Pol Oscillator example

For example, consider a Van der Pol Oscillator with the following equations:

$$\begin{cases}\dot{x}_1 = x_2 \\ \dot{x}_2 = (1-x_1^2)x_2 - x_1\end{cases}, \quad \begin{cases}2 \le x_1 \le 5 \\ -2 \le x_2 \le 2\end{cases} \tag{23}$$

$$\begin{cases}\dot{\hat{x}}_1 = \hat{x}_2 \\ \dot{\hat{x}}_2 = (1+a_1)(1-\hat{x}_1^2)\hat{x}_2 - (1+a_2)\hat{x}_1\end{cases}, \quad \begin{cases}|a_1| \le 0.1 \\ |a_2| \le 0.15\end{cases} \tag{24}$$

This system is simulated using MATLAB and in every δ, the maximum value of $\|\hat{\mathbf{x}}(\delta)-\mathbf{x}(\delta)\|$, called $\|\hat{\mathbf{x}}(\delta)-\mathbf{x}(\delta)\|_{\max}$, and the estimated bound by Gronwall-Bellman Lemma, is plotted in Figure 2. $\|\hat{\mathbf{x}}(\delta)-\mathbf{x}(\delta)\|_{\max}$ is obtained in every δ by changing $\mathbf{x}(0)$ in all over the acceptable domain, expressed in (23), while $\mathbf{x}(0)=\hat{\mathbf{x}}(0)$.

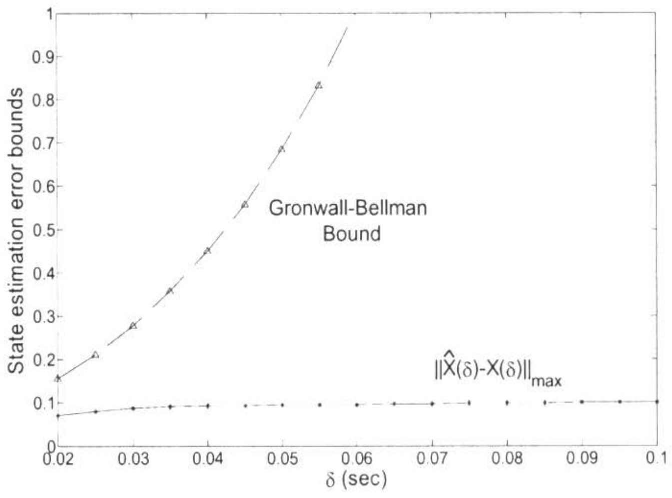


Figure 2- Different estate estimation error bounds vs. estimation time where no noise is considered in measurements for van Der Pol Oscillator example, expressed in (23) and (24).

As shown in Figure 2, the difference between the actual bound and GB bound increases with larger estimation time. The example clearly illustrates the necessity for using other bounds in akin problems. In Figure 3 a linear upper bound is presented. Using polynomials of higher order, sometimes results in having a better estimation, while adding the complexity of the approach.

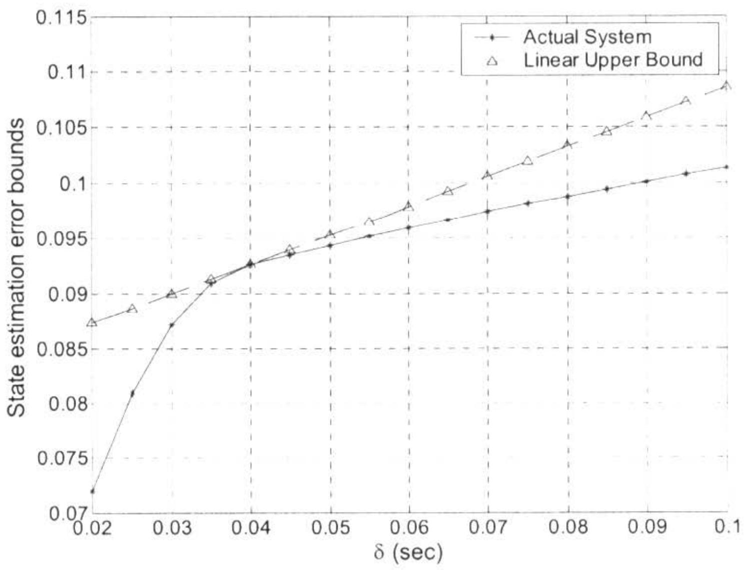


Figure 3- A linear function is used as upper bound for $\left\|\hat{\mathbf{x}}(\delta)-\mathbf{x}(\delta)\right\|_{\max}$, instead of upper bound presented by BG Lemma.

7.3 Scheduling of Multiple Systems with Coupling on a Single Processor

Definition 1. **The set A_i is called the neighbouring set of subsystem *i*, and consists of any subsystem in the network that has direct interconnection with the subsystem *i*.**

Assumption 1. Let us consider the following inequality:

$$\left\|\hat{\mathbf{x}}_i(t+\delta)-\mathbf{x}_i(t+\delta,t)\right\| \le B_i(\delta,\mathbf{x}_i(t),\hat{\mathbf{x}}_i(t)) \tag{25}$$

This means that by knowing the actual and nominal states at time t (i.e., $\hat{\mathbf{x}}_i(t)$ and $\mathbf{x}_i(t)$), the error in estimating the norm of state variables of subsystem i is less than B_i at time $t+\delta$.

Remark 4. As explained in Chapter 7.2, Section 7.2.4, B_i in equation (25) is a function of δ with the state dependent parameters. These parameters are updated at the beginning of the scheduling algorithm and do not change during the operation of scheduling algorithm for finding new execution horizons (periods). Therefore, without loss of generality we consider B_i as a function of δ, i.e. $B_i(\delta)$ in the remaining parts of this report.

Problem Formulation

Consider a network of n dynamically decoupled subsystems using the RHC approach in a decentralized manner. The cost function associated to each subsystem can be expressed as follows:

$$J_i(t,T,\mathbf{x}_i,\mathbf{u}_i,\tilde{\mathbf{x}}_i)=\int_t^{t+T}\left(q_i(\mathbf{x}_i(\tau,t),\mathbf{u}_i(\tau))+\sum_{j\in A_i} g_k(\mathbf{x}_i,\mathbf{x}_j)\right)d\tau \tag{26}$$

where $q_i(\mathbf{x}_i(\tau,t),\mathbf{u}_i(\tau))=\mathbf{x}_i^T(\tau,t)Q_i\mathbf{x}_i(\tau,t)+\mathbf{u}_i^T(\tau)R_i\mathbf{u}_i(\tau)$ and $\tilde{\mathbf{x}}_i$ is a vector containing the states of all neighbouring subsystems of the i^{th} subsystem. g_k is a function which defines the interaction between two nodes of the system (network) $\mathbf{x}_i$ and $\mathbf{x}_j$.

Remark 5. For simplicity of the proof, it supposed that the interaction function g_k is just between two nodes of the system, i.e. $g_k=g_k(\mathbf{x}_i,\mathbf{x}_j)$. After that, the formulation is generalized for any combinations.

As an extension to the (9), the following equation is proposed as an estimation of the closed loop cost in the existence of coupling between subsystems.

$$\bar{J}_{T_2} = \sum_{i=1}^{n}\left(\frac{T_2}{\delta_i}\int^{t+\delta_i}\left(q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}^{*}_{T,i}\left(\tau;\hat{\mathbf{x}}_i(t)\right)\right)+\sum_{j\in A_i} g_k\left(\hat{\mathbf{x}}_i(\tau),\hat{\mathbf{x}}_j(\tau)\right)\right)d\tau\right) \tag{27}$$

Lemma 1. **Consider the following assumptions:**

1- If $\mathbf{x}_i$ and $\mathbf{x}_j$ are two $m\times 1$ column vectors, $\mathbf{x}_{ij}$ is a $2m\times 1$ column vector such that $\mathbf{x}_{ij} = \left[\mathbf{x}_i^{T},\mathbf{x}_j^{T}\right]^{T}$.

2- Let $g_k(\mathbf{x}_{ij})$ be Lipschitz continuous with constant L_g, and $g_k(\mathbf{x}_{ij})$ be a *positive scalar*.

3- From **Assumption 1**, we can have: $\left\|\hat{\mathbf{x}}_i(t+\delta)-\mathbf{x}_i(t+\delta)\right\| \le B_i(\delta)$ and $\left\|\hat{\mathbf{x}}_j(t+\delta)-\mathbf{x}_j(t+\delta)\right\| \le B_j(\delta)$

Show that:

$$\left|g_k\left(\hat{\mathbf{x}}_i(t+\delta),\hat{\mathbf{x}}_j(t+\delta)\right)-g_k\left(\mathbf{x}_i(t+\delta),\mathbf{x}_j(t+\delta)\right)\right| \le L_g\left(B_i+B_j\right) \tag{28}$$

Proof:

From Lipschitz continuity of g_k we have:

$$\left|g_k\left(\hat{\mathbf{x}}_{ij}(t+\delta)\right)-g_k\left(\mathbf{x}_{ij}(t+\delta)\right)\right| \le L_g\left\|\hat{\mathbf{x}}_{ij}(t+\delta)-\mathbf{x}_{ij}(t+\delta)\right\| \tag{29}$$

Furthermore (by omitting $(t+\delta)$ for simplicity):

$$\begin{aligned}&\left\|\hat{\mathbf{x}}_{ij}-\mathbf{x}_{ij}\right\| = \left\|\left[\hat{\mathbf{x}}_i^{T},\hat{\mathbf{x}}_j^{T}\right]^{T}-\left[\mathbf{x}_i^{T},\mathbf{x}_j^{T}\right]^{T}\right\| = \left\|\left[\left(\hat{\mathbf{x}}_i-\mathbf{x}_i\right)^{T},\left(\hat{\mathbf{x}}_j-\mathbf{x}_j\right)^{T}\right]^{T}\right\| =\\ &\left\|\left[\left(\hat{\mathbf{x}}_i-\mathbf{x}_i\right)^{T},\left(\hat{\mathbf{x}}_j-\mathbf{x}_j\right)^{T}\right]\right\| \le \left\|\left(\hat{\mathbf{x}}_i-\mathbf{x}_i\right)^{T}\right\|+\left\|\left(\hat{\mathbf{x}}_j-\mathbf{x}_j\right)^{T}\right\| = \left\|\left(\hat{\mathbf{x}}_i-\mathbf{x}_i\right)\right\|+\left\|\left(\hat{\mathbf{x}}_j-\mathbf{x}_j\right)\right\|\end{aligned} \tag{30}$$

Using the third assumption of this Lemma, results in:

$$\left\|\hat{\mathbf{x}}_{ij}-\mathbf{x}_{ij}\right\| \le B_i + B_j \tag{31}$$

Combination of (29) and (31) results in (28) which completes the proof. □

Corollary 1. **Combination of** Proposition 1 **and** Lemma 1, **results in the following inequality:**

$$\int^{t+\delta_i}\left(q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}^*_{T,i}(\tau;\hat{\mathbf{x}}_i(t))\right)+\sum_{j\in A_i} g_k\left(\hat{\mathbf{x}}_i(\tau),\hat{\mathbf{x}}_j(\tau)\right)\right)d\tau \le$$
$$\int_t^{t+\delta_i} q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right)d\tau+\frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right) \tag{32}$$
$$+\int^{t+\delta_i}\left(\sum_{j\in Ai}\left(g_k\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{x}^*_{T,j}(\tau;\mathbf{x}_j(t))\right)+L_g\left(B_i(\tau-t)+B_j(\tau-t)\right)\right)\right)d\tau$$

Note that the result of corollary 1 is important since it can be used to propose an upper bound on the closed loop cost of (27).

In Proposition 2, the term $q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right)$ was estimated by using the optimal states and inputs resulted from the last open-loop cost calculation and resulted in $q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t-\delta_{p,i})),\mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t-\delta_{p,i}))\right)$. Similarly, the term $g_k(.,.)$ can be estimated using the most recent calculation done on $\mathbf{x}_i$ and $\mathbf{x}_j$. Therefore, the following optimization problem is proposed for scheduling of n dynamically decoupled subsystems in an interconnected network, on a single processor:

$$\min_{\delta_i}\sum_{i=1}^{n}\left(\begin{array}{l}\dfrac{\alpha_i}{\delta_i}\left(\displaystyle\int_t^{t+\delta_i}\left(q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t-\delta_{p,i})),\mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t-\delta_{p,i}))\right)+P_iB_i(\tau-t)\right)d\tau\right)\\ +\dfrac{1}{\delta_i}\displaystyle\int^{t+\delta_i}\left(\sum_{j\in Ai}\beta_k\left(\begin{array}{l}g_k\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t-\delta_{p,i})),\mathbf{x}^*_{T,j}(\tau;\mathbf{x}_j(t-\delta_{p,j}))\right)\\ +L_g\left(B_i(\tau-t)+B_j(\tau-t)\right)\end{array}\right)\right)d\tau\end{array}\right) \tag{33}$$
$$\text{s.t.}\begin{cases}\sum\dfrac{\delta_{c,i}}{\delta_i}\le n(2^{\frac{1}{n}}-1)\\ \delta_{lb,i}<\delta_i\le\delta_{ub,i}\end{cases}$$

If B-G lemma is used for finding B_i, and noise in measurement is considered (like Proposition 1), we would have:

$$B_i(t)=b_{s,i}e^{L_i(t)}+\frac{b_i}{L_i}\left(e^{L_i(t)}-1\right) \tag{34}$$

Therefore, the scheduling optimization problem can be expressed as:

$$\begin{cases} \min\limits_{\delta_i} \sum\limits_{i=1}^{n} \left(\begin{array}{l} \dfrac{\alpha_i}{\delta_i} \left(\begin{array}{l} \int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right),\mathbf{u}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right)\right)d\tau \\ +\dfrac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\dfrac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right) \end{array} \right) \\ +\dfrac{1}{\delta_i}\sum\limits_{j\in Ai}\beta_k \left(\begin{array}{l} \int_t^{t+\delta_i} g_k\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right),\mathbf{x}_{T,j}^*\left(\tau;\mathbf{x}_j\left(t-\delta_{p,j}\right)\right)\right)d\tau \\ +\dfrac{L_gP_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\dfrac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right) \\ +\dfrac{L_gP_j}{L_j}\left(b_{s,j}\left(e^{L_j\delta_i}-1\right)+b_j\left(\dfrac{1}{L_j}\left(e^{L_j\delta_i}-1\right)-\delta_i\right)\right) \end{array} \right) \end{array} \right) \\ \text{s.t.} \begin{cases} \sum \dfrac{\delta_{c,i}}{\delta_i} \le n(2^{\frac{1}{n}}-1) \\ \delta_{lb,i} < \delta_i \le \delta_{ub,i} \end{cases} \end{cases} \tag{35}$$

Remark 6. The formulation, which is presented here, is compatible for decentralized application, since each node has a special cost function. Extension of the present formulation is done in the next chapter, for the cases of having more than one processor, sharing the information via a communication network in the presence of communication delay.

Discussion

The extension of the proposed method can easily be done for having the general case of coupling between the nodes of the network. The key point in the presented method is the proof of Lemma 1, which has two specifications:

1- The Lipschitz continuity of g_k; this specification is not depend on the number of nodes, i.e. for a function g_k which is depend on more than two nodes of the system, the Lipschitz continuity can be defined without any problem.
2- The second specification, which is used in the proof (equation (30)) is actually a general rule that *the norm of a vector is not larger than the summation of norms of its sub-vectors*. Therefore, this specification is valid for the general case of g_k as well.

Therefore, the proposed method is general enough for dynamic scheduling of multiple systems having interconnection in their controller cost function, on a single processor. For example, the formation of multiple vehicles using a single processor is one of the cases that this method can be applied.

In addition, the case that is studied in this chapter is important because as stated before, by generalization of this method, the case of multiple subsystems on multiple computers can be done. This case is studied in the next chapter.

7.4 Computation and Communication Scheduling of Multiple Systems on Multiple Distributed Processors

The problem under study is illustrated in Figure 4. For this case, we consider delay in communication and we determine execution horizon and communication period for each subsystem.

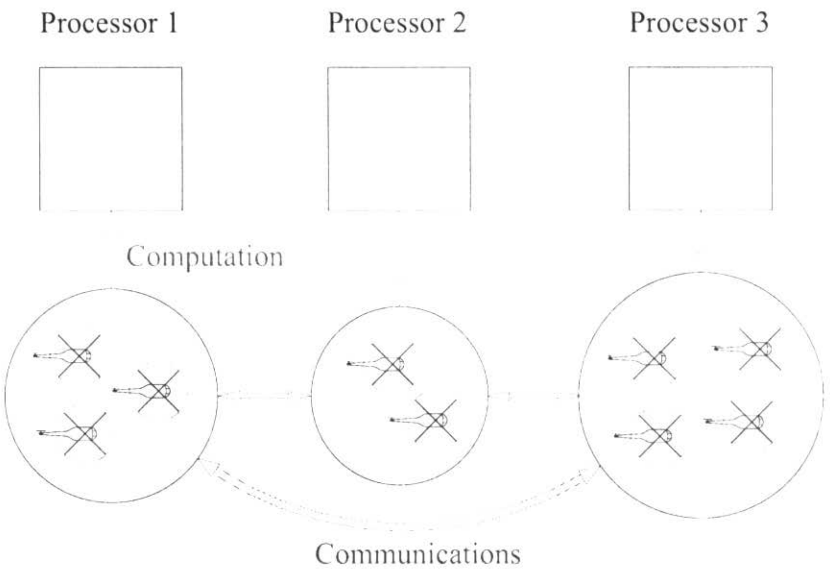


Figure 4- Illustration of Computation and Communication Scheduling for the case of three processors and presented configuration.

7.4.1 Problem Statement

Suppose having two computers connected via a local network and these computers have n_1 and n_2 number of subsystems using RHC approach, respectively. The total number of systems is n_v. Therefore, we have: $n_v = n_1 + n_2$. The method is later generalized for more than two computers.

Assumption 2: Communication period should be $\zeta_i = z_i \delta_i$ where ζ_i is the communication period of system *i*, δ_i is the execution horizon, and z_i is an integer.

Remark 7: Maximum communication delay of the data sent from node *i* to node *j*, is equal to:
$\tau_{ij} = \zeta_i + \xi_{ij}$
where ξ_{ij} is the delay because of senders and receivers and this can be considered as a fixed (known) delay if the hopping between nodes *i* and *j* does not change.

It is supposed that the data sent from subsystem *j*, has delay equal to τ_{ij}, when it is received in subsystem *i*. Therefore, the open-loop cost of subsystem *i*, can be expressed as follows:

$$J_i(t,T,\mathbf{x}_i,\mathbf{u}_i,\widetilde{\mathbf{x}}_i)=\int_t^{t+T}\left(q_i(\mathbf{x}_i(\tau;t),\mathbf{u}_i(\tau))+\sum_{j\in A_i}g_k(\mathbf{x}_i(\tau;t),\mathbf{x}_j(\tau;t-\tau_{ij}))\right)d\tau \quad (36)$$

Like equation (9) in chapter 7.2, we propose the following cost as an estimation of the closed loop cost:

$$\bar{J}_{T_2}=\sum_{i=1}^{n}\left(\frac{T_2}{\delta_i}\int_t^{t+\delta_i}\left(q_i(\hat{\mathbf{x}}_i(\tau),\mathbf{u}^*_{T,i}(\tau;\hat{\mathbf{x}}_i(t)))+\sum_{j\in A_i}g_k(\hat{\mathbf{x}}_i(\tau),\hat{\mathbf{x}}_j(\tau))\right)d\tau\right) \quad (37)$$

Similar to what we have done in previous chapters, we are going to find an upper bound on equation 37. Before considering this, we present a Lemma which considers the effect of communication delay.

Lemma 2. Consider the following assumptions:

1- If $\mathbf{x}_i$ and $\mathbf{x}_j$ are two $m\times 1$ column vectors, $\mathbf{x}_{ij}$ is a $2m\times 1$ column vector such that $\mathbf{x}_{ij}=\left[\mathbf{x}_i^T,\mathbf{x}_j^T\right]^T$.

2- Let $y_k(\mathbf{x}_{ij})=g_k(\mathbf{x}_i,\mathbf{x}_j)$ be Lipschitz continuous with constant L_g, and $y_k(\mathbf{x}_{ij})$ be a *positive scalar.*

3- From **Assumption 1** and **Remark 4**, we can have: $\left\|\hat{\mathbf{x}}_i(t+\delta)-\mathbf{x}_i(t+\delta,t)\right\|\le B_i(\delta)$.

4- The data which is received in node *i* from node *j*, has a delay of τ_{ij}.

Show that:

$$g_k(\hat{\mathbf{x}}_i(t+\delta),\hat{\mathbf{x}}_j(t+\delta))\le g_k(\mathbf{x}_i(t+\delta,t),\mathbf{x}_j(t+\delta,t-\tau_{ij}))+L_g(B_i(\delta)+B_j(\delta+\tau_{ij}))\quad (38)$$

Proof:
From Lipschitz continuity of y_k we have:

$$\begin{aligned}&\left|y_k(\hat{\mathbf{x}}_{ij}(t+\delta))-y_k\left(\left[\mathbf{x}_i(t+\delta,t)^T,\mathbf{x}_j(t+\delta,t-\zeta_{ij})^T\right]^T\right)\right|\le\\&L_g\left\|\hat{\mathbf{x}}_{ij}(t+\delta)-\left[\mathbf{x}_i(t+\delta,t)^T,\mathbf{x}_j(t+\delta,t-\zeta_{ij})^T\right]^T\right\|\end{aligned}\quad (39)$$

As $x\le|x|$, we can have:

$$y_k\left(\hat{\mathbf{x}}_{ij}(t+\delta)\right) \le y_k\left(\left[\mathbf{x}_i(t+\delta,t)^T, \mathbf{x}_j\left(t+\delta,t-\zeta_{ij}\right)^T\right]^T\right) + \\ L_g\left\|\hat{\mathbf{x}}_{ij}(t+\delta) - \left[\mathbf{x}_i(t+\delta,t)^T, \mathbf{x}_j\left(t+\delta,t-\zeta_{ij}\right)^T\right]^T\right\| \tag{40}$$

Furthermore:

$$\left\|\hat{\mathbf{x}}_{ij}(t+\delta) - \left[\mathbf{x}_i(t+\delta,t)^T, \mathbf{x}_j\left(t+\delta,t-\zeta_{ij}\right)^T\right]^T\right\| \le \\ \left\|\hat{\mathbf{x}}_i(t+\delta) - \mathbf{x}_i(t+\delta,t)\right\| + \left\|\hat{\mathbf{x}}_j(t+\delta) - \mathbf{x}_j\left(t+\delta,t-\zeta_{ij}\right)\right\| \tag{41}$$

Using the third assumption of this Lemma, results in:

$$\left\|\hat{\mathbf{x}}_j(t+\delta) - \mathbf{x}_j\left(t+\delta,t-\zeta_{ij}\right)\right\| \le B_j\left(\delta+\zeta_{ij}\right) \tag{42}$$

Therefore, combination of (41) and (42) results in:

$$\left\|\hat{\mathbf{x}}_{ij}(t+\delta) - \left[\mathbf{x}_i(t+\delta,t)^T, \mathbf{x}_j\left(t+\delta,t-\zeta_{ij}\right)^T\right]^T\right\| \le B_i(\delta) + B_j\left(\delta+\zeta_{ij}\right) \tag{43}$$

Combination of (40) and (43) results in (38) which completes the proof. □

Corollary 2. Similar to what we expressed in Corollary 1, combination of Proposition 1 and Lemma 2, results in the following inequality in presence of communication delay:

$$\int_t^{t+\delta_i}\left(q_i\left(\hat{\mathbf{x}}_i(\tau), \mathbf{u}^*_{T,i}(\tau;\hat{\mathbf{x}}_i(t))\right) + \sum_{j\in A_i} g_k\left(\hat{\mathbf{x}}_i(\tau), \hat{\mathbf{x}}_j(\tau)\right)\right)d\tau \le \\ \int_t^{t+\delta_i}\left(q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)), \mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right) + P_i B_i(\tau-t)\right)d\tau \\ + \int_t^{t+\delta_i}\sum_{j\in A_i}\begin{pmatrix} g_k\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)), \mathbf{x}^*_{T,j}\left(\tau;\mathbf{x}_j(t_p)\right)\right) \\ + L_g\left(B_i(\tau-t) + B_j\left(\tau-t+\tau_{ij}\right)\right)\end{pmatrix}d\tau \tag{44}$$

where, $\mathbf{x}^*_{T,j}\left(\tau;\mathbf{x}_j(t_p)\right)$ indicates the optimal trajectory of subsystem j, resulted from sampling point t_p (See *Appendix A* to have better understanding about communication delay and t_p). In addition, we have two cases for τ_{ij} (See **Assumption 2**, and **Remark 7**):

$$\begin{cases} \tau_{ij} = \delta_j \text{ , } if \ \ i \ and \ \ j \ are \ in \ the \ same \ computer \\ \tau_{ij} = z_j\delta_j + \xi_{ij} \text{ , } if \ \ i \ and \ \ j \ are \ in \ different \ computers \end{cases}$$

Remark 8: It should be noted that for decentralized application, this is valid for the case that we are using the produced trajectory by the neighbours, i.e., in order to find optimal inputs of subsystem *i*, the optimal trajectories that have been estimated by the neighbours are considered [13].

Based on the result of **Corollary 2**, the following optimization problem is proposed for computation and communication scheduling of n_v subsystems on two computers, by having the assumption that $\zeta_i = z_i\delta_i$: (computation scheduling means finding the proper values for δ_i and the communication scheduling means finding the proper communication rate (or period) for each subsystem. The combination of these two problems leads to finding δ_i and z_i.)

$$\min_{\delta_i, z_i}\begin{cases}\sum_{i=1}^{n_1} G_i \Rightarrow CPU1\\ \sum_{i=n_1+1}^{n_1+n_2} G_i \Rightarrow CPU2\end{cases}$$

$$s.t.\begin{cases}\sum_{i=1}^{n_j}\frac{\delta_{c,i}}{\delta_i}\le n_j(2^{\frac{1}{n_j}}-1)\ \ (j=1,2)\\ \sum_{i=1}^{n_v}\frac{\mathbf{B}_i}{z_i\delta_i}\le BW\end{cases} \tag{45}$$

where $\mathbf{B}_i$ is the amount of Bytes to be sent in each communication of system *i*, and *BW* is the *part* of network Bandwidth that we want to use. G_i is defined from the following (same as the previous scheduling cost):

$$G_i = \begin{pmatrix}\frac{\alpha_i}{\delta_i}\left(\int_t^{t+\delta_i}\left(q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right)+P_iB_i(\tau-t)\right)d\tau\right)\\ +\frac{1}{\delta_i}\int^{t+\delta_i}\left(\sum_{j\in A_i}\beta_k\begin{pmatrix}g_k\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{x}^*_{T,j}(\tau;\mathbf{x}_j(t_p))\right)\\ +L_g\left(B_i(\tau-t)+B_j(\tau-t+z_j\delta_j+\xi_{ij})\right)\end{pmatrix}\right)d\tau\end{pmatrix} \tag{46}$$

If B-G lemma is used for finding B_i, and noise in measurement is considered (**like Proposition 1**), we would have:

$$B_i(t)=b_{s,i}e^{L_it}+\frac{b_i}{L_i}\left(e^{L_it}-1\right) \tag{47}$$

Combination of (46) and (47) by assuming no noise in measurement (i.e., $b_{s,i} = 0$), **results in:**

$$G_i = \left(\begin{array}{l} \dfrac{\alpha_i}{\delta_i} \left(\displaystyle\int_t^{t+\delta_i} \left(q_i\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{u}^*_{T,i}(\tau;\mathbf{x}_i(t))\right) + \dfrac{P_i b_i}{L_i}\left(\dfrac{e^{L_i\delta_i}-1}{L_i} - \delta_i\right)\right) d\tau \right) \\ + \dfrac{1}{\delta_i} \displaystyle\sum_{j \in A_i} \beta_k \left(\begin{array}{l} \displaystyle\int^{t+\delta_i} g_k\left(\mathbf{x}^*_{T,i}(\tau;\mathbf{x}_i(t)),\mathbf{x}^*_{T,j}(\tau;\mathbf{x}_j(t_p))\right) d\tau \\ + \dfrac{L_g b_i}{L_i}\left(\dfrac{e^{L_i\delta_i}-1}{L_i} - \delta_i\right) + \dfrac{L_g b_j}{L_j}\left(\dfrac{e^{L_j(z_j\delta_j+\xi_{ij})}}{L_j}\left(e^{L_j\delta_i}-1\right) - \delta_i\right) \end{array} \right) \end{array} \right) \quad (48)$$

Remark 9: It should be noted that in the formulation of (45) it is supposed that any subsystem needs to communicate with other processor, i.e., it has a neighbour on that processor.

Remark 10: The term BW is equation (45), is a *part* of the available bandwidth and we cannot use the whole bandwidth of the network. Similar to the criteria on the usage of CPU, when we use Rate Monotonic Priority, it is possible to find the criteria on communication. However, in the case of communication, we deal with non-preemptive tasks and some assumptions should be considered in the length of messages.

7.4.2 Extension to More than Two Processors

Generalization of the approach to the cases of having more than two processors is done in the following.
Extension of the algorithm is almost straightforward except the constraint on the communication bandwidth. See Figure (4) as an example of more than two processors. In this example, subsystems 1, 2, and 3 are in processor 1, subsystems 4 and 5 are in processor 2, and subsystems 6, 7, 8, and 9 are in processor 3. If the neighbours of subsystem 2 for example are 1, 7, and 8, then the information of subsystem 2 should be sent to processor 3 only. This example, help us in the following definition.

Definition 2: C_i is the set of processors that have at least one neighbour of subsystem i. Note that the processor having subsystem i on it, is not considered in C_i.

Using the definition of C_i, the constraint on communication bandwidth can be defined as follows:

$$\sum_{i=1}^{n_v} \sum_{j \in C_i} \frac{\mathbf{B}_i}{z_i \delta_i} \le BW \quad (49)$$

7.5 Application to Hovercraft Problem

In this Section, the simulation results of applying proposed scheduling method in Section 7.2, on multiple hovercraft systems is investigated.

For the hovercraft example, the upper bound presented by Gronwall-Bellman (GB), was close to the upper bound resulted from the actual system simulation. Therefore, the bound that is used in this example, is GB and for dynamic scheduling, the parameters in GB (*b* and *L*) should be updated.

7.5.1 Hovercraft Model

A complete discussion on the modeling of the RC hovercraft is presented in [9]. Since only viscous friction is considered in the nominal model, the model is similar to that presented in [10]. From the results of [9], it is straightforward to assume a linear relationship between the applied voltage and the thrust produced by the propellers. Therefore, based on Figure 5, the nominal model expressed in local coordinate is given as:

$$\begin{aligned}
\dot{u} &= \frac{1}{M}\left(a_r V_r + a_l V_l - b_1 u\right) + v r \\
\dot{v} &= -\frac{1}{M} b_2 v - u r \\
\dot{r} &= \frac{l}{2J}\left(a_l V_l - a_r V_r\right) - \frac{1}{J} b_3 r
\end{aligned} \tag{50}$$

where M is the mass, J is the moment of inertia about the Z-axis, and b_1, b_2 and b_3 are the coefficients of viscous friction in X, Y, and rotational directions, respectively. Also, V_r and V_l are the applied voltages to the right and left propeller DC motors, respectively.

Another model of the RC hovercraft with more details is considered as the actual model of the system and used in simulations. This model is as follows [9]:

$$\begin{aligned}
\dot{\omega}_{p,r} &= \frac{1}{J_a}\left(\frac{k_{e2}}{R}\left(V_r - k_{e1}\omega_{p,r}(t)\right) - k_{a3}\omega_{p,r}{}^2 - k_{a4}\omega_{p,r}(u - \frac{l}{2}r)\right) \\
\dot{\omega}_{p,l} &= \frac{1}{J_a}\left(\frac{k_{e2}}{R}\left(V_l - k_{e1}\omega_{p,l}(t)\right) - k_{a3}\omega_{p,l}{}^2 - k_{a4}\omega_{p,r}(u + \frac{l}{2}r)\right) \\
\dot{u} &= \frac{1}{M}\left(k_{a1}\omega_{p,r}{}^2 - k_{a2}\omega_{p,r}\left(u - \frac{l}{2}r\right) + k_{a1}\omega_{p,l}{}^2 - k_{a2}\omega_{p,l}\left(u + \frac{l}{2}r\right) - f_{f,u}\right) - g\sin(\theta) + vr \\
\dot{v} &= -\frac{f_{f,v}}{M} - g\sin(\phi) - ur \\
\dot{r} &= \frac{l}{2J}\left(\left(k_{a1}\omega_{p,l}{}^2 - k_{a2}\omega_{p,l}\left(u + \frac{l}{2}r\right)\right) - \left(k_{a1}\omega_{p,r}{}^2 - k_{a2}\omega_{p,r}\left(u - \frac{l}{2}r\right)\right)\right) - \frac{f_{f,r}}{J}
\end{aligned} \tag{51}$$

where J_a is the moment of inertia of propeller and rotating part of the DC motor, $\omega_{p,r}$ and $\omega_{p,l}$ are the angular speed of the right and left propellers, respectively. θ and ϕ are the pitch and roll angels of hovercraft indicating the ground slop. $f_{f,u}$, $f_{f,v}$ and $f_{f,r}$ are the friction functions presented in [9].

This model includes actuator dynamics, propeller dynamics, effect of hovercraft speed on the thrust produced by propeller, nonlinearities in friction applied in hovercraft, and the ground inclination. In addition, noises in measurements are applied in simulations.

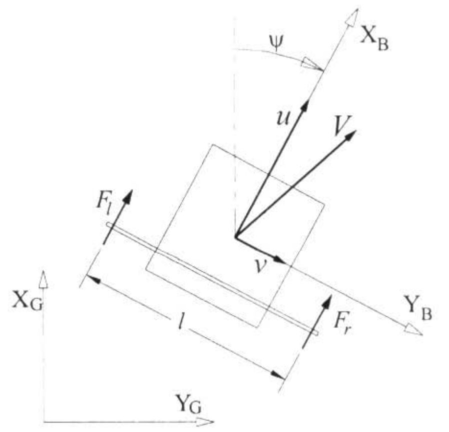


Figure 5- RC hovercraft model with produced thrusts by propellers and local (body) and global coordinate systems

7.5.2 Simulation Results

The proposed scheduling algorithm in chapter 7.2 is applied in the following simulation. This is done by applying the method two hovercrafts. These hovercrafts are following similar circular paths, and have different levels of uncertainty. Subsystem 2 is the hovercraft with more uncertainty in its model. Two cases are considered: static scheduling which the execution horizons of two systems does not change during process, and dynamic scheduling using the proposed cost function and algorithm. The simulations are performed using an Intel Pentium IV processor with 2.66 GHz speed, Microsoft Visual C++ 6.0, Venturcom RTX 6.0.1, and the RHC Object Oriented Library (RHCOOL)[33].

For the RC hovercraft example with model parameters presented in [11], the Gronwall-Bellman bound is close to the system response [11]. Therefore, the cost function presented in (18) is used in this example. In addition, the scheduling parameters (P_i , b_i and L_i) are found offline for different values of states. Since the state variables *u*, *v*, and *r* are the only variables that have effects on the scheduling parameters, only these states are used to find the scheduling parameters. These parameters are stored in different tables and are used in on-line simulations.

The paths followed by two systems in static scheduling and dynamic scheduling cases are presented in Figure 6 and Figure 7, respectively. In the first case, the periods of two subsystems are selected based on the worst-case computation time and are 0.4 and 0.31 seconds for subsystem 1 and 2, respectively.

As illustrated in these two figures, in dynamic scheduling case, both hovercrafts were able to follow their trajectories better than the static scheduling case.

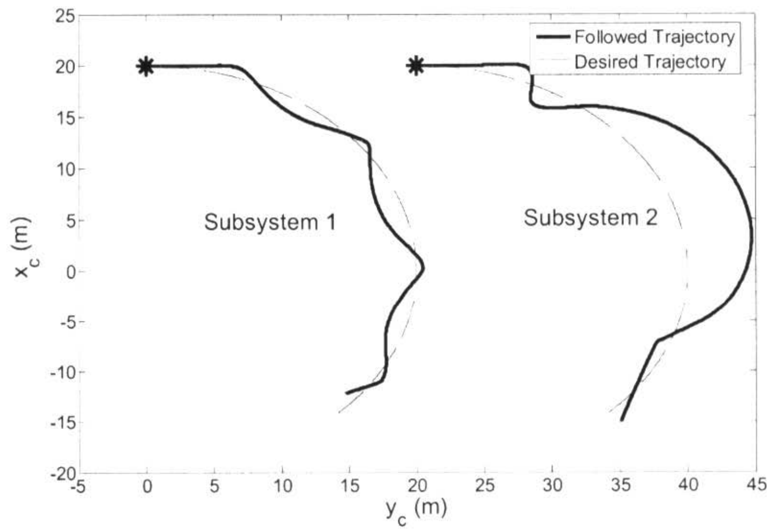


Figure 6- Paths followed by two hovercrafts in static scheduling. Execution horizon of Subsystem 1 is 0.4 second and for Subsystem 2 is 0.31 seconds. The bolded lines are representing the position of the center of each hovercraft.

[33] This library was developed recently in the CIS Lab, Concordia University. For more information, see [12], Section 3.3.

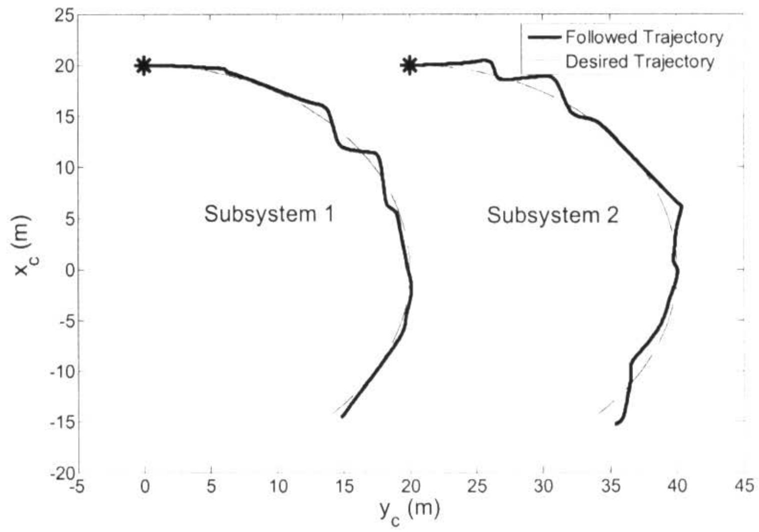


Figure 7- Paths followed by two hovercrafts in dynamic scheduling using the proposed algorithm

In order to have better feeling about the performance of the presented cases, the position errors of both subsystems regarding their reference trajectories are demonstrated in Figure 8 and Figure 9.

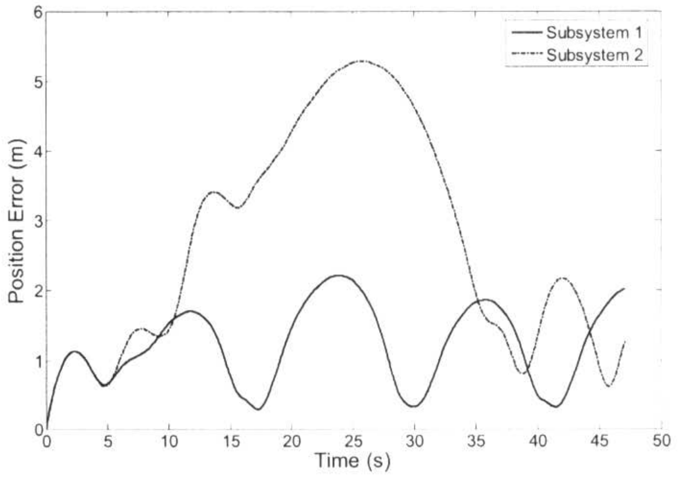


Figure 8- Error in trajectory following for both subsystems in static scheduling case

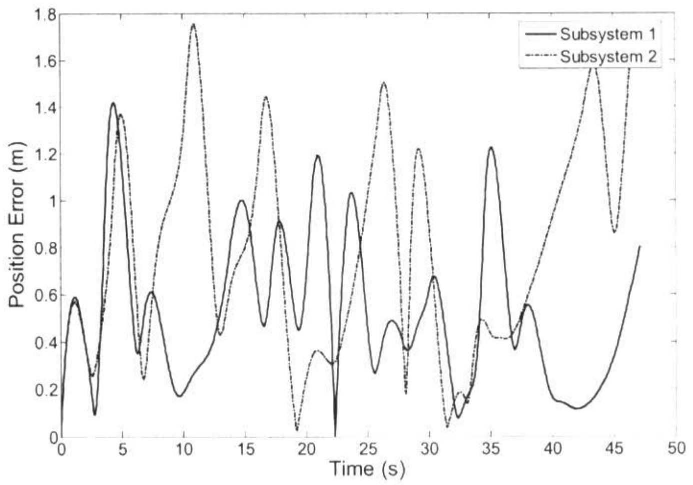


Figure 9- Error in trajectory following for both subsystems in dynamic scheduling case

Figure 10 shows the changing in execution horizons of both subsystems in dynamic scheduling case. Since, T_2 was selected 1 second in this case; the execution horizons remain unchanged for about 1 second. In addition, the results are presented in shorter period for more clarity of the diagram.

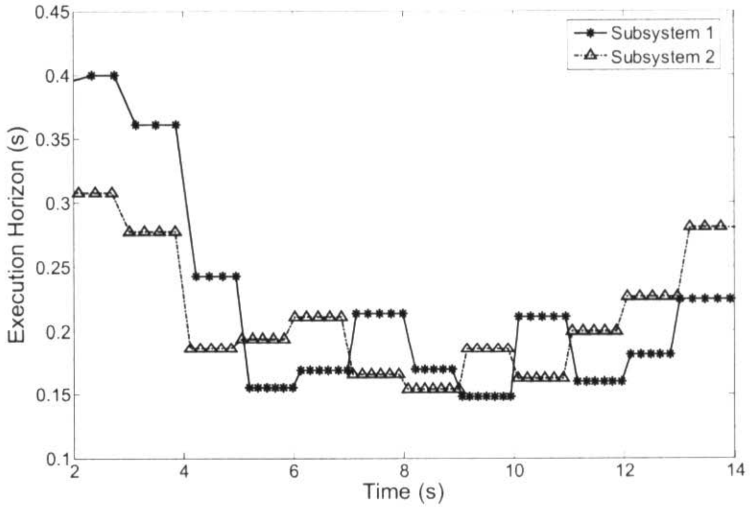


Figure 10- Changing of execution horizon vs. time in dynamic scheduling case

7.6 Conclusion and Future Works

In this report, a new method was developed for dynamic scheduling of multiple RHC systems. The problems studied in this report, have nonlinear dynamics subject to uncertainties in the model and sensor noise. The formulation presented in this report was developed based on a bound from the Gronwall-Bellman Lemma. This lemma calculates a conservative bound in some nonlinear problems and should be replaced by less conservative bounds in highly nonlinear systems. The proposed algorithm was applied to multiple RC hovercrafts simulations to illustrate the new approach.

This report was explaining Task #11 of the first proposal.

7.7 References

[1] Mayne, D. Q., Rawlings, J. B., Rao, C. V., Scokaert, P. O. M., "Constrained model predictive control: Stability and optimality", *Automatica*, 36, 2000, pp. 789-814.

[2] H. Chen, F. Allgower, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability", *Automatica*, 34, 1998, pp.1205-1217.

[3] H. Kopetz, "Real-time systems: Design principles for distributed embedded applications", Chapters 9 and 11, *Springer*, 1997.

[4] H. Khalil, "Nonlinear Systems", Prentice Hall, 2002.

[5] D. Henriksson, J. Akesson, "Flexible implementation of model predictive controller using sub-optimal solutions", Technical Report, Department of Automatic Control, Lund Institute of Technology, Sweden, 2004.

[6] D. Henriksson, A. Cervin, "Optimal On-line Sampling Period Assignment for Real-Time Control Tasks Based on Plant State Information", Proc. of the 44th IEEE Conf. on Decision and Control, and the European Control Conf., 2005, pp. 4469-4474.

[7] A. Jadbabaie, "Receding horizon control of nonlinear systems: A control Lyapunov function approach", *PhD Thesis, California Institute of Technology*, Pasadena, CA, 2000.

[8] M. B. Milam, R. Franz, J. E. Hauser, R. M. Murray, "Receding Horizon Control of a Vectored Thrust Flight Experiment", *IEE Proceedings on Control Theory and Applications*, 152(3), 2005, pp. 340-348.

[9] A. Azimi, B. W. Gordon, "Modeling of RC Hovercraft", Technical Report, CIS Lab, Concordia University, July 2006.

[10] A. P. Aguiar, L. Cremean, J.P. Hespanha, "Position tracking for a nonlinear underactuated hovercraft: controller design and experimental results", Proc. of the 42nd Conf. on Decision and Control, 2003, pp. 3858-3863.

[11] A. Azimi, B. W. Gordon, "discussion on Gronwall-Belmann Lemma for van Der Pol Oscillator and RC hovercraft problem", Technical Report, CIS Lab, Concordia University, September 2006.

[12] A. Azimi, B. Gholami, B. W. Gordon, "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques: Real-time Scheduling of Multiple Uncertain Receding Horizon Control Systems", Progress Report, CIS Lab, Concordia University, March 2006.

[13] T. Keviczky, "Decentralized Receding Horizon Control of Large Scale Dynamically Decoupled Systems", *Ph.D. Thesis, Control Science and Dynamical Systems Center, University of Minnesota,* September 2005.

[14] A. Azimi, B.W. Gordon, C.A. Rabbath, "Dynamic Scheduling of Decentralized Receding Horizon Controllers on Concurrent Processors for the Cooperative Control of Unmanned Systems", Submitted to 46th IEEE Conference on Decision and Control (CDC 2007).

7.8 Appendix A: A Discussion on Communication Delay

The delay term τ_{ij} which considered in (36) can vary from ξ_{ij} to $\xi_{ij}+\zeta_i$ where ξ_{ij} is the delay in sender and receiver, and ζ_i is the communication period of subsystem *i*. Since in the scheduling algorithm, we find an upper bound and optimize that bound, the maximum value of that delay is considered in the scheduling optimization problem (45). In the following, this varying delay is described.

Consider Figure 1. The data, which are receiving in subsystem *i* from subsystem *j*, can arrive at any time and two ultimate cases are shown in Figure 1-b. Suppose having no computation delay.

Case 1) If the data arrives just a little before sampling time t ($t-\varepsilon$), the updated data of *j* will be used in calculation of subsystem *i*. Therefore, the data from *j* have a delay equal to $\xi_{ij}+\varepsilon$. Therefore, when ε goes to zero ($\varepsilon \to 0$), we have the least delay, i.e. $\mathbf{x}_j\left(\tau;t-\xi_{ij}\right)$ and it is the most *lucky case.*

Case 2) The second case is when the data is arrived just a little after sampling time t ($t+\varepsilon$). In this case, last available data from *j* will be used in *i*. Therefore, when $\varepsilon \to 0$ we have $\mathbf{x}_j\left(\tau;t-\zeta_j-\xi_{ij}\right)$ which is the worst case.

In summary, in the case that we have communication delay, but no computation delay we can express the trajectory of $\mathbf{x}_j$ as $\mathbf{x}_j\left(\tau;t-\tau_{ij}\right)$ where:

$$\begin{cases} \tau_j=\delta_j \text{ , } \mathit{if} \;\; i \;\; \mathit{and} \;\; j \;\; \mathit{are\ in\ the\ same\ computer} \\ \tau_j=z_j\delta_j+\xi_j \text{ , } \mathit{if} \;\; i \;\; \mathit{and} \;\; j \;\; \mathit{are\ in\ different\ computers} \end{cases}$$

Remark 11:
The term $\zeta_j=z_j\delta_j$ is just the communication period and even if we are simulating on one computer, this term can occur, however in the case of having j and i on the same computer, $z_j=1$. In addition, It is for the case that we are using the produced trajectory by the neighbours (i.e., subsystem *j*), while we are calculating the inputs and trajectories of system *i*.

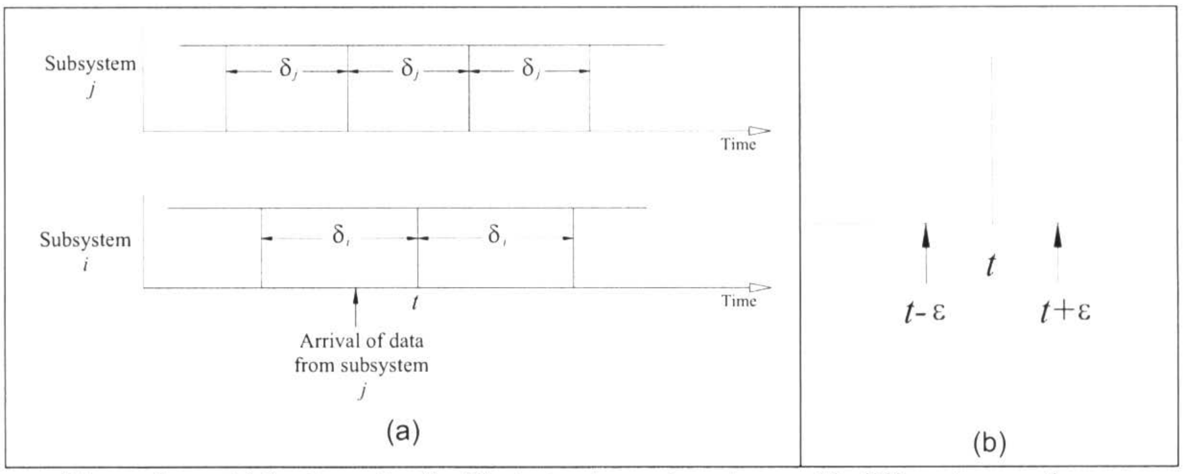


Figure 1- part (a) schematically illustrates two subsystems with different execution horizons having communication. Part (b) indicates two ultimate cases, Lucky case ($t-\varepsilon$) and worst case ($t+\varepsilon$)

CHAPTER 8: REAL-TIME SCHEDULING OF MULTIPLE UNCERTAIN RECEDING HORIZON CONTROL SYSTEMS RUNNING ON DISTRIBUTED COMPUTERS

(Tasks #12, 13, and 14)

Ali Azimi
PhD Student &
Research Assistant

Yan Zhao
Msc Student &
Research Assistant

Brandon W. Gordon
Assistant Professor

Control and Information Systems (CIS) Laboratory
Department of Mechanical and Industrial Engineering
Concordia University
Montreal, Quebec, Canada H3G 1M8

March 2007

Abstract

This report documents the work related to tasks #12, #13, and #14 for the project entitled "Synthesis and Implementation of Single - and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques". This report briefly describes the design of a real time scheduling algorithm for computation and communication scheduling of multiple uncertain receding horizon control systems, running on multiple distributed computers. Verification of the proposed method has been performed using a set of 3DOF miniature rotorcrafts. This demonstrates the performance and feasibility of the proposed method. The performance of the method is also demonstrated by simulating it in Visual C++ by using RHC Object Oriented Library (RHCOOL), which is adapted for Dynamic Scheduling of Decentralized RHC Systems.

Introduction

Scheduling of RHC computational and communication is very important for decentralized RHC control since it allows optimization of resources to achieve maximum performance in the presence of uncertainty, disturbances, and delays. The tasks addressed in this report related to the project entitled "Real-time Scheduling of Multiple Uncertain Receding Horizon Control Systems" are as follows:

Task 12: Validate RHC performance via Matlab simulations, and distributed hard real-time simulations running on a parallel cluster of digital processors.

Task 13: Test RHC and scheduling algorithms on multi-rotorcraft experimental test-bed.

Task 14: Documentation and explanation of software and results

For these tasks, we designed a scalable, dynamic run-time scheduling algorithm for multiple RHC running on distributed digital processors (task #11) that has been delivered in the previous report. In this work, dynamic scheduling algorithms for both computation and communication scheduling of multiple processors are developed that directly optimize robust RHC performance of the whole network. This is achieved through minimization of the upper bound on the so-called *overall closed-loop cost* of the system, subject to scheduling constraints on both computation and communication.

Feasibility and performance of the proposed scheduling methods was demonstrated through C++ real-time simulations running on a parallel cluster of digital processors. It was done by having different sets of simulations and scenarios:

a) The performance of distributed RHC was demonstrated by using multiple miniature helicopters in the formation with trajectory generation and obstacle avoidance. Dynamic formation was also demonstrated briefly (task #12).
b) The proposed scheduling methods were examined with two sets of simulation; first, by using two computers each one having two hovercrafts while maintaining their formation; second by using four hovercrafts with four processors (task #12).
c) Distributed RHC simulations with varying communications and … (Hojjat)

In addition, the performance of the RHC method was demonstrated by applying it on the miniature helicopters. Furthermore, the foresaid scheduling methods are applied on some sets of 3DOF miniature under actuated rotorcrafts. These rotorcrafts are the modified version of the miniature underactuated hovercrafts. It was done by using four hovercrafts with four processors. In both tests, overhead vision system is used to obtain necessary information for feedback (task #13).

Finally, all of the programs and main functions used to obtain the foresaid results, were gathered together for easy accessing. All of the programs and results are stored on the complementary CDs, and they are explained in the Appendix (Task #14).

The following chapters are included in this report:

In chapter two, a scalable, dynamic run-time scheduling algorithm for multiple RHC running on distributed digital processors that has been developed recently and explained in [2] is explained briefly. In this chapter, a new dynamic scheduling approach is proposed that considers the effect of modeling uncertainty for multiple continuous time RHC systems. This is accomplished by combining a scheduling approach with results from continuous time nonlinear systems theory. Assuming the prediction horizon and communication period are two variables, using a limited computing, and communicating resource, a new technique is proposed for determining the execution horizon and communication period for multiple distributed RHC systems. The problem of determining the execution horizon for each subsystem while optimizing the performance is cast into a constrained optimization problem.

In chapter three, application to 3DOF under-actuated miniature hovercraft problem has been investigated. Since RHC approach is a model-based algorithm, modeling and identification of the apparatus has been done, briefly. In addition, using the RHCOOL, the dynamic scheduling of multiple hovercrafts has been investigated based on the designed scheduling method. In this investigation, sets of two and four processors are being used to control four hovercrafts with RHC method, while the computation and communication is being scheduled using the proposed dynamic scheduling approach.

In chapter four, experimental results have been reported for miniature helicopter and hovercrafts. The results are from implementing RHC approach to single helicopter as well as scheduling results of multiple processors (two and four) with four hovercrafts.

Conclusions and future work are presented in chapter five and references are in chapter six.

Chapter 2 of this report is describing [2] in brief, chapter 3 is describing Task #12, and chapter 4 of the report is describing Task #13 of the first proposal. In addition, Appendix A is explaining the materials on the complementary CDs (Task #14).

8.1 Dynamic Scheduling Algorithm for Multiple RHC Running on Distributed Digital Processors

This chapter briefly describes the dynamic scheduling approach recently developed at the CIS lab. Part of the result was recently published on [1], while the whole chapter described in [2].

8.1.1 Scheduling of Multiple RHC Systems without Coupling on a Single Processor

Consider the problem of controlling n decoupled uncertain nonlinear systems using the RHC scheme. The following equations serve as the nominal model for describing the decoupled plants

$$\dot{\mathbf{x}}_i = \mathbf{f}_i\left(\mathbf{x}_i(t), \mathbf{u}_i(t), t\right),\ i = 1, \ldots, n \tag{1}$$

in addition, the actual plants are described by

$$\dot{\hat{\mathbf{x}}}_i = \mathbf{f}_i\left(\hat{\mathbf{x}}_i(t), \mathbf{u}_i(t), t\right) + \mathbf{g}_i(\hat{\mathbf{x}}_i, \mathbf{u}_i, t),\ i = 1, \ldots, n \tag{2}$$

For the proposed approach, the execution horizons of all subsystems should be defined such that the overall performance of the system is maximized. In order to evaluate the performance of the system, the following cost function is proposed as the cost of the closed loop system from time t to $t + T_2$, where T_2 is the period that calculated execution horizons would be applied to

$$\hat{J}_{T_2} = \sum_{i=1}^{n}\left(\sum_{k=1}^{d_i}\left(\int_{t_k^i}^{t_k^i+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau), \mathbf{u}_{T,i}^*\left(\tau; \hat{\mathbf{x}}_i\left(t_k^i\right)\right)\right) d\tau\right) + \int_{t+d_i\delta_i}^{t+T_2} q_i\left(\hat{\mathbf{x}}_i(\tau), \mathbf{u}_{T,i}^*\left(\tau; \hat{\mathbf{x}}_i\left(t + d_i\delta_i\right)\right)\right) d\tau\right) \tag{3}$$

where $d_i = \lfloor T_2 / \delta_i \rfloor$[34], $t_k^i = t + (k-1)\delta_i$, and $\mathbf{u}_{T,i}^*\left(\tau; \hat{\mathbf{x}}_i\left(t_k^i\right)\right)$ is the optimal input applied to subsystem *i*.

The idea is to find the execution horizon of each subsystem (δ_i), such that $\sum_{i=1}^{n} \hat{J}_{T_2}^i$ is minimum. As explained in [2], instead of calculating $\hat{J}_{T_2}$, the following cost is proposed which is an estimation of the cost in (7):

[34] Floor function of a real number *x*, denoted $\lfloor x \rfloor$, is the largest integer less than or equal to *x*.

$$\bar{J}_{T_2}(\hat{\mathbf{x}}(t),\mathbf{u}_T^*(.)) = \sum_{i=1}^{n}\left(\frac{T_2}{\delta_i}\int_t^{t+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}_{T,i}^*(\tau;\hat{\mathbf{x}}_i(t))\right) d\tau\right) \tag{4}$$

Proposition1:

Suppose the following assumptions hold true:

5. q_i is quadratic, so: $q_i(\mathbf{x}_i,\mathbf{u}_i) = \mathbf{x}_i^T Q_i \mathbf{x}_i + \mathbf{u}_i^T R_i \mathbf{u}_i$
6. $\overline{Q}_i(\mathbf{x}_i) = \mathbf{x}_i^T Q_i \mathbf{x}_i$ is Lipschitz continuous with constant P_i.
7. $\mathbf{f}_i$ in equation (5) is Lipschitz continuous with constant L_i.
8. $\mathbf{g}_i$ in equation (6) is bounded and $\|\mathbf{g}_i(\mathbf{x}_i,\mathbf{u}_i)\| \le b_i$.

then the following inequality exists:

$$\int_t^{t+\delta_i} q_i\left(\hat{\mathbf{x}}_i(\tau),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right) d\tau \le$$
$$\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*(\tau;\mathbf{x}_i(t)),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t))\right) d\tau + \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right) \tag{5}$$

where $b_{s,i}$ is the bound on measurement errors caused by sensor noise.

Proof: **See [2].**

Proposition 2:

Consider n decoupled systems with equations presented in (5) and (6), controlled by RHC using only one computer. In addition, the assumptions of proposition 1 are valid. Since uncertainties in the subsystems are different and the measurements are performed with bounded sensor noise, show that the following equation can be used to optimally determine the execution horizons of all subsystems:

$$\min_{\delta_i}\sum_{i=1}^{n}\frac{\alpha_i}{\delta_i}\left(\begin{array}{l}\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*(\tau;\mathbf{x}_i(t-\delta_{p,i})),\mathbf{u}_{T,i}^*(\tau;\mathbf{x}_i(t-\delta_{p,i}))\right)d\tau\\ +\frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i}-1\right)+b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i}-1\right)-\delta_i\right)\right)\end{array}\right)$$
$$\text{s.t.}\quad\begin{cases}\sum\frac{\delta_{c,i}}{\delta_i}\le n(2^{\frac{1}{n}}-1)\\ \delta_{lb,i}<\delta_i\le\delta_{ub,i}\end{cases} \tag{6}$$

where $\delta_{ub,i}$ and $\delta_{ub,i}$ are the maximum and minimum acceptable execution horizons for subsystem *i*, respectively. $\delta_{p,i}$ is the execution horizon of subsystem *i* before being updated.

Proof: **See [2]**

8.1.2 Scheduling of Multiple Systems with Coupling Effect on a Single Processor

Definition 1. The set A_i is called the neighboring set of subsystem *i*, and consists of any subsystem in the network that has direct interconnection with the subsystem *i*.

Assumption 1. Let us consider the following inequality:

$$\left\|\hat{\mathbf{x}}_i(t+\delta)-\mathbf{x}_i(t+\delta,t)\right\| \le B_i\left(\delta,\mathbf{x}_i(t),\hat{\mathbf{x}}_i(t)\right) \tag{7}$$

This means that by knowing the actual and nominal states at time t (i.e., $\hat{\mathbf{x}}_i(t)$ and $\mathbf{x}_i(t)$), the error in estimating the norm of state variables of subsystem i is less than B_i at time $t+\delta$.

Remark 1. As explained in [2], B_i in equation (25) is a function of δ with the state dependent parameters. These parameters are updated at the beginning of the scheduling algorithm and do not change during the operation of scheduling algorithm for finding new execution horizons (periods). Therefore, without loss of generality we consider B_i as a function of δ, i.e. $B_i(\delta)$ in the remaining parts of this section.

Problem Formulation

Consider a network of n dynamically decoupled subsystems using the RHC approach in a decentralized manner. The cost function associated to each subsystem can be expressed as follows:

$$J_i\left(t,T,\mathbf{x}_i,\mathbf{u}_i,\widetilde{\mathbf{x}}_i\right)=\int_t^{t+T}\left(q_i\left(\mathbf{x}_i(\tau,t),\mathbf{u}_i(\tau)\right)+\sum_{j\in A_i} g_k\left(\mathbf{x}_i,\mathbf{x}_j\right)\right)d\tau \tag{8}$$

where $q_i\left(\mathbf{x}_i(\tau,t),\mathbf{u}_i(\tau)\right)=\mathbf{x}_i^T(\tau,t)Q_i\mathbf{x}_i(\tau,t)+\mathbf{u}_i^T(\tau)R_i\mathbf{u}_i(\tau)$ and $\widetilde{\mathbf{x}}_i$ is a vector containing the states of all neighboring subsystems of the i^{th} subsystem. g_k is a function which defines the interaction between two nodes of the system (network) $\mathbf{x}_i$ and $\mathbf{x}_j$.

Remark 1. For simplicity of the proof, it supposed that the interaction function g_k is just between two nodes of the system, i.e. $g_k = g_k(\mathbf{x}_i, \mathbf{x}_j)$. After that, the formulation is generalized for any combinations.

As an extension to the (9), the following equation is proposed as an estimation of the closed loop cost in the existence of coupling between subsystems.

$$\bar{J}_{T_2} = \sum_{i=1}^{n} \left(\frac{T_2}{\delta_i} \int_t^{t+\delta_i} \left(q_i\left(\hat{\mathbf{x}}_i(\tau), \mathbf{u}^*_{T,i}(\tau; \hat{\mathbf{x}}_i(t))\right) + \sum_{j \in A_i} g_k\left(\hat{\mathbf{x}}_i(\tau), \hat{\mathbf{x}}_j(\tau)\right) \right) d\tau \right) \tag{9}$$

Lemma 1. Consider the following assumptions:

4- If $\mathbf{x}_i$ and $\mathbf{x}_j$ are two $m \times 1$ column vectors, $\mathbf{x}_{ij}$ is a $2m \times 1$ column vector such that $\mathbf{x}_{ij} = \left[\mathbf{x}_i^T, \mathbf{x}_j^T\right]^T$.

5- Let $g_k(\mathbf{x}_{ij})$ be Lipschitz continuous with constant L_g, and $g_k(\mathbf{x}_{ij})$ be a *positive scalar*.

6- From **Assumption 1**, we can have: $\left\|\hat{\mathbf{x}}_i(t+\delta) - \mathbf{x}_i(t+\delta)\right\| \le B_i(\delta)$ and $\left\|\hat{\mathbf{x}}_j(t+\delta) - \mathbf{x}_j(t+\delta)\right\| \le B_j(\delta)$

Show that:

$$\left| g_k\left(\hat{\mathbf{x}}_i(t+\delta), \hat{\mathbf{x}}_j(t+\delta)\right) - g_k\left(\mathbf{x}_i(t+\delta), \mathbf{x}_j(t+\delta)\right) \right| \le L_g\left(B_i + B_j\right) \tag{10}$$

Proof: See [2].

Corollary 1. **Combination of** Proposition 1 **and** Lemma 1, **results in the following inequality:**

$$\begin{aligned} &\int_t^{t+\delta_i} \left(q_i\left(\hat{\mathbf{x}}_i(\tau), \mathbf{u}^*_{T,i}(\tau; \hat{\mathbf{x}}_i(t))\right) + \sum_{j \in A_i} g_k\left(\hat{\mathbf{x}}_i(\tau), \hat{\mathbf{x}}_j(\tau)\right) \right) d\tau \le \\ &\int_t^{t+\delta_i} q_i\left(\mathbf{x}^*_{T,i}(\tau; \mathbf{x}_i(t)), \mathbf{u}^*_{T,i}(\tau; \mathbf{x}_i(t))\right) d\tau + \frac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i} - 1\right) + b_i\left(\frac{1}{L_i}\left(e^{L_i\delta_i} - 1\right) - \delta_i \right) \right) \\ &+ \int_t^{t+\delta_i} \left(\sum_{j \in Ai} \left(g_k\left(\mathbf{x}^*_{T,i}(\tau; \mathbf{x}_i(t)), \mathbf{x}^*_{T,j}(\tau; \mathbf{x}_j(t))\right) + L_g\left(B_i(\tau - t) + B_j(\tau - t)\right) \right) \right) d\tau \end{aligned} \tag{11}$$

The following optimization problem is proposed for scheduling of *n* dynamically decoupled subsystems in an interconnected network, on a single processor:

$$\min_{\delta_i} \sum_{i=1}^{n} \left(\begin{array}{l} \dfrac{\alpha_i}{\delta_i}\left(\displaystyle\int_t^{t+\delta_i} \left(q_i\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right), \mathbf{u}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right)\right) + P_i B_i(\tau - t)\right) d\tau \right) \\ + \dfrac{1}{\delta_i} \displaystyle\int_t^{t+\delta_i} \left(\sum_{j\in Ai} \beta_k \left(\begin{array}{l} g_k\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right), \mathbf{x}_{T,j}^*\left(\tau;\mathbf{x}_j\left(t-\delta_{p,j}\right)\right)\right) \\ + L_g\left(B_i(\tau-t) + B_j(\tau-t)\right) \end{array} \right) \right) d\tau \end{array} \right) \tag{12}$$

$$\text{s.t.} \begin{cases} \sum \dfrac{\delta_{c,i}}{\delta_i} \le n(2^{\frac{1}{n}} - 1) \\ \delta_{lb,i} < \delta_i \le \delta_{ub,i} \end{cases}$$

If B-G lemma is used for finding B_i, and noise in measurement is considered (like Proposition 1 **), we would have:**

$$B_i = b_{s,i} e^{L_i(\tau - t)} + \frac{b_i}{L_i}\left(e^{L_i(\tau - t)} - 1\right) \tag{13}$$

Therefore, the scheduling optimization problem can be expressed as:

$$\begin{cases} \min\limits_{\delta_i} \displaystyle\sum_{i=1}^{n} \left(\begin{array}{l} \dfrac{\alpha_i}{\delta_i} \left(\begin{array}{l} \displaystyle\int_t^{t+\delta_i} q_i\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right), \mathbf{u}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right)\right) d\tau \\ + \dfrac{P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i} - 1\right) + b_i\left(\dfrac{1}{L_i}\left(e^{L_i\delta_i} - 1\right) - \delta_i\right)\right) \end{array} \right) \\ + \dfrac{1}{\delta_i} \displaystyle\sum_{j\in Ai} \beta_k \left(\begin{array}{l} \displaystyle\int_t^{t+\delta_i} g_k\left(\mathbf{x}_{T,i}^*\left(\tau;\mathbf{x}_i\left(t-\delta_{p,i}\right)\right), \mathbf{x}_{T,j}^*\left(\tau;\mathbf{x}_j\left(t-\delta_{p,j}\right)\right)\right) d\tau \\ + \dfrac{L_g P_i}{L_i}\left(b_{s,i}\left(e^{L_i\delta_i} - 1\right) + b_i\left(\dfrac{1}{L_i}\left(e^{L_i\delta_i} - 1\right) - \delta_i\right)\right) \\ + \dfrac{L_g P_j}{L_j}\left(b_{s,j}\left(e^{L_j\delta_i} - 1\right) + b_j\left(\dfrac{1}{L_j}\left(e^{L_j\delta_i} - 1\right) - \delta_i\right)\right) \end{array} \right) \end{array} \right) \\ \text{s.t.} \begin{cases} \sum \dfrac{\delta_{c,i}}{\delta_i} \le n(2^{\frac{1}{n}} - 1) \\ \delta_{lb,i} < \delta_i \le \delta_{ub,i} \end{cases} \end{cases} \tag{14}$$

Remark 1. **The formulation, which is presented here, is compatible for decentralized application, since each node has a special cost function. Extension of the present formulation will be done for the cases of having more than one processor, sharing**

the information via a communication network.

8.1.3 Computation and Communication Scheduling of Multiple Systems on Multiple Distributed Processors

The problem under study is illustrated in Figure 4. For this case, we consider delay in communication and we determine execution horizon and communication period for each subsystem.

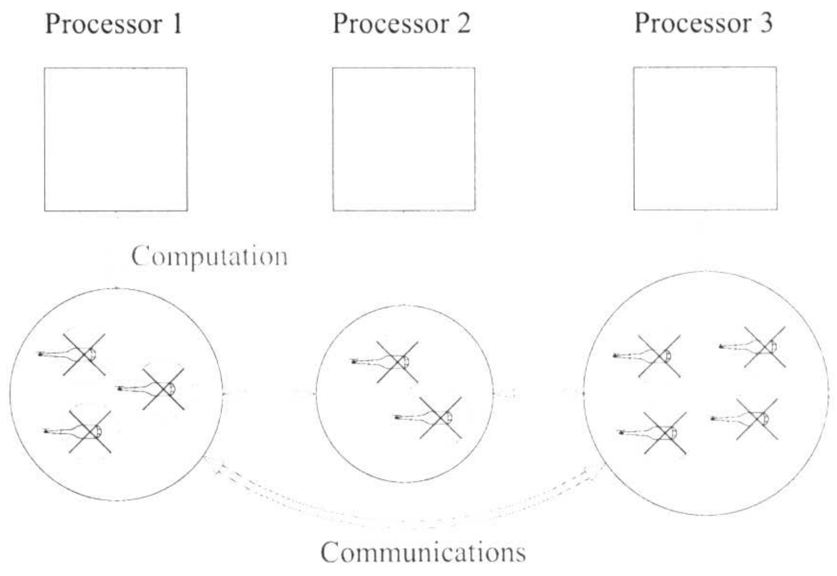


Figure 1- Illustration of Computation and Communication Scheduling for the case of three processors and presented configuration.

Suppose having two computers connected via a local network and these computers have n_1 and n_2 number of subsystems using RHC approach, respectively. The total number of systems are n_v. Therefore, we have: $n_v = n_1 + n_2$. The method is later generalized for more than two computers.

Remark 1: Communication period should be $\zeta_i = z_i \delta_i$ where ζ_i is the communication period of system *i*, δ_i is the execution horizon, and z_i is an integer.

Remark 2: Maximum communication delay of the data sent from node *i* to node *j*, is equal to:

$\tau_{ij} = \zeta_i + \xi_{ij}$

where ξ_{ij} is the delay because of senders and receivers and this can be considered as a fixed (known) delay.

When the data sent from subsystem *j*, has delay equal to τ_{ij}, the open-loop cost of subsystem i, can be expressed as follows:

$$J_i(t,T,\mathbf{x}_i,\mathbf{u}_i,\widetilde{\mathbf{x}}_i)=\int_t^{t+T}\left(q_i(\mathbf{x}_i(\tau;t),\mathbf{u}_i(\tau))+\sum_{j\in A_i}g_k(\mathbf{x}_i(\tau;t),\mathbf{x}_j(\tau;t-\tau_{ij}))\right)d\tau \quad (15)$$

Like before (equation (9)), we propose the following cost as an estimation of the closed loop cost:

$$\bar{J}_{T_2}=\sum_{i=1}^{n}\left(\frac{T_2}{\delta_i}\int_t^{t+\delta_i}\left(q_i(\hat{\mathbf{x}}_i(\tau),\mathbf{u}^*_{T,i}(\tau;\hat{\mathbf{x}}_i(t)))+\sum_{j\in A_i}g_k(\hat{\mathbf{x}}_i(\tau),\hat{\mathbf{x}}_j(\tau))\right)d\tau\right) \quad (16)$$

Based on the worst case analysis, for the open loop cost function, we have (note that we are using the produced trajectory of other subsystems instead of estimating the trajectory of other subsystems):

$$\sum_{i=1}^{n}\left(\frac{T_2}{\delta_i}\int_t^{t+\delta_i}\left(q_i(\mathbf{x}^*_i(\tau;t),\mathbf{u}^*_{T,i}(\tau;t))+\sum_{j\in A_i}g_k(\mathbf{x}^*_i(\tau;t),\mathbf{x}^*_j(\tau;t-\tau_{ij}))\right)d\tau\right) \quad (17)$$

where we have two cases for τ_{ij}:

$$\begin{cases}\tau_{ij}=\delta_j \text{ , } \mathit{if\ i\ and\ j\ are\ in\ the\ same\ computer}\\ \tau_{ij}=z_j\delta_j+\xi_{ij} \text{ , } \mathit{if\ i\ and\ j\ are\ in\ different\ computers}\end{cases}$$

Remark: It should be noted that this is valid for the case that we are using the produced trajectory by the neighbors.

The following optimization problem is proposed for computation and communication scheduling of n_v subsystems on two computers, by having the assumption that $\zeta_i=z_i\delta_i$: (computation scheduling means finding the proper values for δ_i and the communication scheduling means finding the proper communication rate (or period) for each subsystem. The combination of these two problems leads to finding δ_i and z_i.)

$$\min_{\delta_i, z_i} \begin{cases} \sum_{i=1}^{n_1} G_i \Rightarrow CPU1 \\ \sum_{i=n_1+1}^{n_1+n_2} G_i \Rightarrow CPU2 \end{cases}$$

$$s.t. \begin{cases} \sum_{i=1}^{n_j} \frac{\delta_{c,i}}{\delta_i} \le n_j (2^{\frac{1}{n_j}} - 1) \quad (j=1,2) \\ \sum_{i=1}^{n_v} \frac{\mathbf{B}_i}{z_i \delta_i} \le BW \end{cases} \tag{18}$$

where $\mathbf{B}_i$ is the amount of Bytes to be sent in each communication of system i, and BW is the Bandwidth of the network. G_i is defined from the following (same as the previous scheduling cost):

$$G_i = \left(\begin{array}{l} \frac{\alpha_i}{\delta_i} \left(\int_t^{t+\delta_i} \left(q_i \left(\mathbf{x}^*_{T,i}(\tau; \mathbf{x}_i(t)), \mathbf{u}^*_{T,i}(\tau; \mathbf{x}_i(t)) \right) + P_i B_i(\tau - t) \right) d\tau \right) \\ + \frac{1}{\delta_i} \int^{+\delta_i} \left(\sum_{j \in A_i} \beta_k \left(\begin{array}{l} g_k \left(\mathbf{x}^*_{T,i}(\tau; \mathbf{x}_i(t)), \mathbf{x}^*_{T,j}(\tau; \mathbf{x}_j(t_p)) \right) \\ + L_g \left(B_i(\tau - t) + B_j(\tau - t + z_j \delta_j + \xi_{ij}) \right) \end{array} \right) \right) d\tau \end{array} \right) \tag{19}$$

If B-G lemma is used for finding B_i, and noise in measurement is considered (**like Proposition 1**), we would have:

$$B_i(t) = b_{s,i} e^{L_i t} + \frac{b_i}{L_i} \left(e^{L_i t} - 1 \right) \tag{20}$$

Combination of (46) and (20) by assuming no noise in measurement (i.e., $b_{s,i} = 0$**), results in:**

$$G_i = \left(\begin{array}{l} \frac{\alpha_i}{\delta_i} \left(\int_t^{t+\delta_i} \left(q_i \left(\mathbf{x}^*_{T,i}(\tau; \mathbf{x}_i(t)), \mathbf{u}^*_{T,i}(\tau; \mathbf{x}_i(t)) \right) + \frac{P_i b_i}{L_i} \left(\frac{e^{L_i \delta_i} - 1}{L_i} - \delta_i \right) \right) d\tau \right) \\ + \frac{1}{\delta_i} \sum_{j \in A_i} \beta_k \left(\begin{array}{l} \int^{+\delta_i} g_k \left(\mathbf{x}^*_{T,i}(\tau; \mathbf{x}_i(t)), \mathbf{x}^*_{T,j}(\tau; \mathbf{x}_j(t_p)) \right) d\tau \\ + \frac{L_g b_i}{L_i} \left(\frac{e^{L_i \delta_i} - 1}{L_i} - \delta_i \right) + \frac{L_g b_j}{L_j} \left(\frac{e^{L_j (z_j \delta_j + \xi_{ij})}}{L_j} \left(e^{L_j \delta_i} - 1 \right) - \delta_i \right) \end{array} \right) \end{array} \right) \tag{21}$$

Generalization of the approach to the cases of having more than two processors is done in the following.
Extension of the presented approach is almost straightforward except the constraint on the communication bandwidth. The following definition helps us in this matter.

Definition 2: C_i is the set of processors that have at least one neighbor of subsystem *i*. Note that the processor having subsystem *i* on it, is not considered in C_i.

Using the definition of C_i, the constraint on communication bandwidth can be defined as follows:

$$\sum_{i=1}^{n_v}\sum_{j\in C_i}\frac{\mathbf{B}_i}{z_i\delta_i}\le BW \tag{22}$$

8.2 Distributed Real-time Simulations with Application to 3DOF Miniature Helicopter and Hovercraft Problem

In the following two chapters, first the distributed RHC performance is studied by applying the method to multiple miniature helicopter systems. Secondly, the proposed dynamic scheduling approach is applied to multiple hovercraft systems on distributed computers.

8.2.1 Distributed RHC of Multi-Vehicle Systems

This chapter briefly describes the distributed Receding Horizon controllers of multi helicopter systems recently developed at the CIS lab [4].
In this section, distributed RHC with three vehicles are investigated. The problem is formation of three vehicles for some leader-follower examples. The examples are consists of real-time simulations with obstacle avoidance and dynamic formation.

A battlefield scenario is set up for the distributed receding horizon control of three helicopters chasing one target, avoiding one obstacle while keeping a triangular formation. Different cost functions are applied for each helicopter. Preliminary simulation results of dynamic formation control, dynamically adding and reducing helicopters and dynamically adding and reducing targets are also presented in this section.

8.2.1.1 Problem Setup and Proposed Method of Solution

RHCOOL [3], Receding Horizon Control Object-Oriented Library, is used for this simulation. This library aims at precisely and conveniently simulating RHC control of a model or models by declaring different parameters and cost functions for each model. Since in this application, the helicopters are assumed to be same in dynamics but different in their individual tasks, different cost functions are developed for each vehicle in the target following, obstacle avoidance and formation maintenance.

Three computers and a gigabit fast Ethernet switch are used for the distributed simulation. In addition, UDP protocol is applied for data transmission in order to guarantee the high-speed data exchange between those systems. However, the whole process is not in real time manner, but in a sequential manner to prevent any possible data lost. In this section, the simulations are run on both one computer and on three computers in a distributed manner, and the results are compared.

In [42], the concept of agent brigade is applied to robotic soccer team. This aspire us using a subsystem concept in the dynamic formation control simulation. A group of helicopters chasing one target is considered as one individual system. Each helicopter within this system is divided into subsystem performing its individual task. The simulation is run on one computer, on which each helicopter control process is operated sequentially in a non real time manner. Two targets are put into battlefield one after anther.

The task is divided into two major parts, a path planner, which chases the target and avoids the obstacle on the way, and two followers that catch up with the planner and keep a certain distance from each other.

In [40], a choice of cost function for the formation of multiple UAVs is introduced. Moreover, for the cost function of path planner in this simulation, which is responsible for tracking the target and doing the killing, is shown below.

$$J_1(k) = J_1^{st}(\mathbf{x}_1(k)) + J_1^{moa}(\mathbf{x}_1(k)) + J_1^{slow}(\mathbf{x}_1(k)) \tag{23}$$

where

$$J_1^{st}(k) = \frac{1}{2}(\mathbf{y}_d(k) - \mathbf{x}_1(k))^T Q(\mathbf{y}_d(k) - \mathbf{x}_1(k)) \tag{24}$$

$J_1^{st}(\mathbf{x}_1(k))$ is used for target tracking the target, $\mathbf{y}_d(k)$ is the position data of the target at time k, and $\mathbf{x}_1(k)$ is the position data of the path planner at time k

In this chasing scenario, an obstacle will be on the way of the three helicopters formation and they need to avoid colliding with it. In [41], a potential function is introduced into the cost function of RHC controller for the obstacle avoidance task. $J_1^{moa}(k)$ in the cost function is used for the obstacle avoidance,

$$J_1^{moa}(k) = \frac{K}{(\sqrt{(x_1(k) - x_o)^2 + (y_1(k) - y_o)^2} - RR)^2} \tag{25}$$

where $(x_1(k), y_1(k))$ designate the position data of the path planner at time k, while (x_o, y_o) is the coordinate data of the obstacle, and RR is the distance between the center of the obstacle and the path planner.

Even though the combination of $J_1^{st}(\mathbf{x}_1(k))$ and $J_1^{moa}(k)$ is enough for trajectory following, only this two part will not give us satisfying result of simulation. Because no matter how large the cost will be, RHC controller will give the leader a huge input to decrease the cost to zero, an input that is too huge to be implemented in reality and that is able to make the helicopter "jump" to the target at time $k + 1$ from where it is at time k.

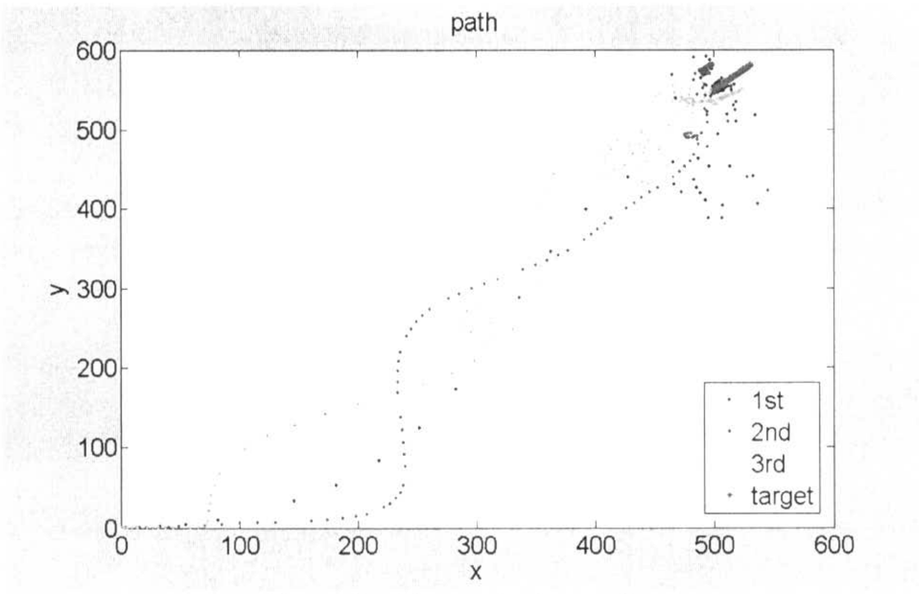


Figure 2- without velocity penalty

Therefore, another function $J_1^{slow}(k)$ to slow down the leader is needed in the cost function.

$$J_1^{slow}(k) = \beta_1(u_1(k)^2 + v_1(k)^2) \tag{26}$$

$J_1^{slow}(k)$ is a penalty is the cost function for the path planner, where $u_1(k)$ and $v_1(k)$ are the axial velocity and radial velocity of the path planner respectively. After adding this part to the cost function, the path of the leader becomes satisfying.

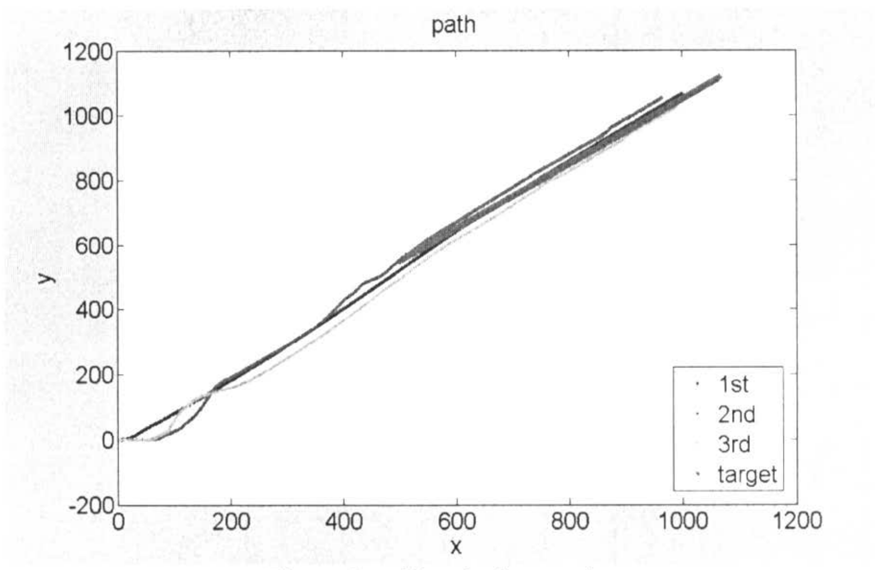


Figure 3- with velocity penalty

For the followers in this system, there is another set of cost functions for each of them.

$$J_i = J_i^{fmj}(\mathbf{x}(k)) + J_i^{slow}(\mathbf{x}(k)) + J_i^{moc}(\mathbf{x}(k)) \qquad (27)$$

$$J_i^{fmj} = \underset{i\neq j, i\neq 1}{\alpha}{}^{ij}_{p}\left[\sqrt{(x^i - x^j)^2 + (y^i - y^j)^2} - R\right]^2 \qquad (28)$$

J_i^{fmj} is responsible for i-th follower to keep the formation, (x_i, y_i) is the position data of the i-th vehicle, while (x_j, y_j) is the data position of other j vehicles. R is used to keep the distance between each two helicopters.

$$J_i^{slow} = \beta_i(u_i(k)^2 + v_i(k)^2) \qquad (29)$$

Same as the path planner, each i-th helicopter needs a penalty function to slow down the velocity of them.

For each i-th vehicle, there is another function to protect each helicopter would not crash into each other. Even though the J_i^{fmj} function is responsible and doing a fairly good job on keeping formation, there is still high possibility for some of the vehicles crashing into each other when the path planner is moving too fast.

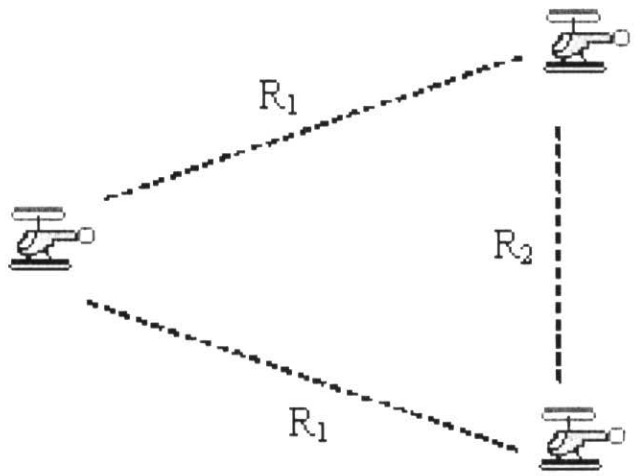


Figure 4- helicopter formation

As in Figure 4, ideally, R_1 and R_2 should be equal with the length of the triangle formation R. However, the leader is set to follow the target and avoid obstacle only, which means formation is not in its consideration. Therefore, when leader is far away from the target, it tends to move faster since its cost function is large; on the other hand, the followers are not able to accelerate until their distance from the leader R_1 becomes larger. This delay will cause $J_i^{fm1}(k)$ for the followers increase significantly and dominate the whole cost function, while $\underset{i\neq 1, i\neq j}{J_i^{fmj}(k)}$ has weaker influence, so the distance between the followers R_2 could not be maintained. The followers would only collide with each other, since R_2 is impossible to increase because of the optimization of the cost function.

Therefore another function J_i^{moc} should be introduced into the cost function for the followers.

$$J_i^{moc} = \frac{K}{\left(\sqrt{\left(x_i(k)-x_j(k)\right)^2+\left(y_i(k)-y_j(k)\right)^2}-r_{safe}\right)^2} \qquad (30)$$

where r_{safe} is the safe radius for the followers. r_{safe} is a relatively small number, small enough for not interfering the formation function, but larger than the diameter of helicopter so that they would not collide. In this simulation, r_{safe} is set to 5m. Even though in Figure 3 there is a crossing between these two helicopters, they actually avoid that collision, as shown in Figure 5, which reflects the distance between the two followers.

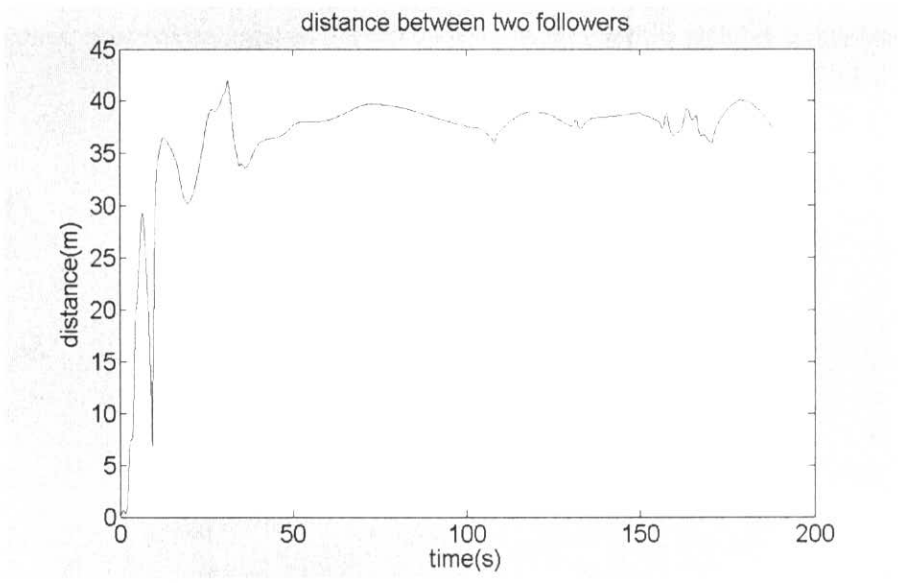


Figure 5- Distance between two followers

8.2.1.2 Distributed Target Following and Obstacle Avoidance Simulation

In Figure 6, the structure of distributed simulation is illustrated, and the procedure is followed after that.

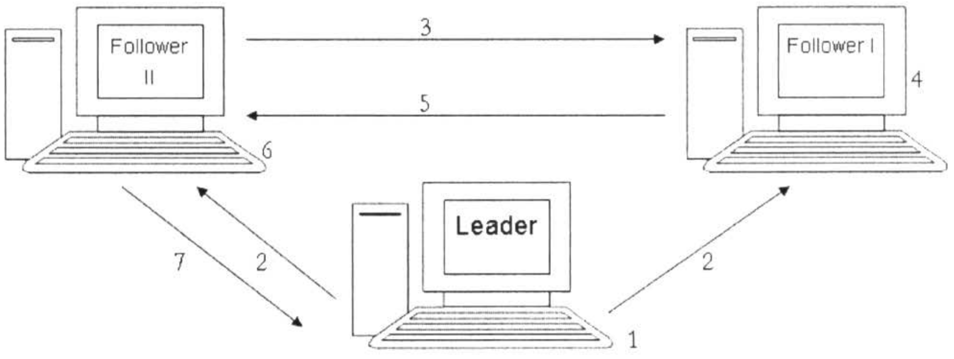


Figure 6- Three computers distributed simulation

1. Calculate inputs and next position of leader based on the target position data generated
2. Send leader's position to each follower
3. Send follower II's position
4. Calculate inputs and next position of follower I based on the data received from leader and follower II
5. Send follower I's position
6. Calculate inputs and next position of follower II based on the data received from leader and follower I
7. Send positions of follower I and II to the leader
8. Go back to 1

The simulation is in sequential manner, which means one task will take place only if the task ahead of it is finished to make sure no or least data lost due to timing of data transfer in the network. However, in the section (3.2) the simulations are done using the proposed real-time scheduler for data communication.

In Figure 7, the scenario is that the target vehicle makes a turn, while the group of the three helicopters avoids the round obstacle when chasing it. In Figure 8, scenario is more complicated. The target is moving in a circle path, and the group of the helicopters avoids the round obstacle twice on its way of chasing the target.

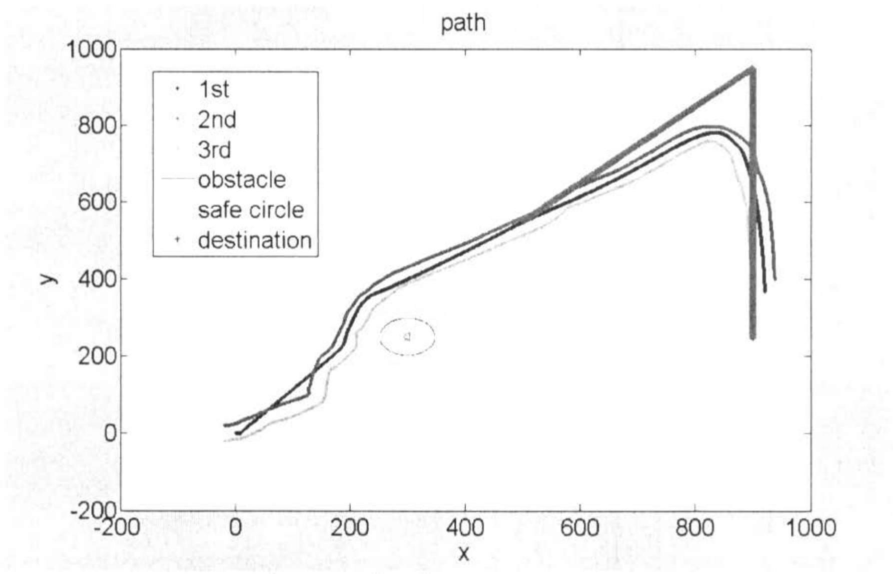


Figure 7- distributed simulation 01

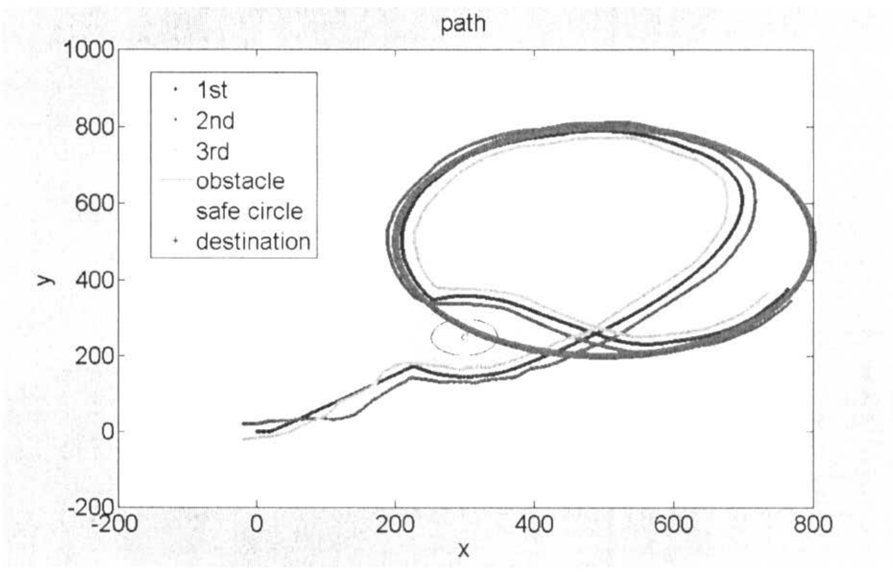


Figure 8- distributed simulation 02

8.2.1.3 Dynamic Formation

Subsystem concept is used in the dynamic formation control simulation. A group of helicopters chasing one target is considered as one individual system. Each helicopter within this system is divided into subsystem performing its individual task.

In this section, a group of six helicopters, first, forms a triangular formation and chases a target, which is moving far away. Then at time t_k another target shows up in the scenario. The six helicopters will divide into two separate groups, three vehicles for each group, and chase the two targets separately.

Each group contains one path planner as the leader, and two followers to form a smaller triangular team. The cost functions for each leader and follower could be in the same way of the cost functions as discussed in part 2 of this paper, although some values of the parameters might need to be changed for better performance.

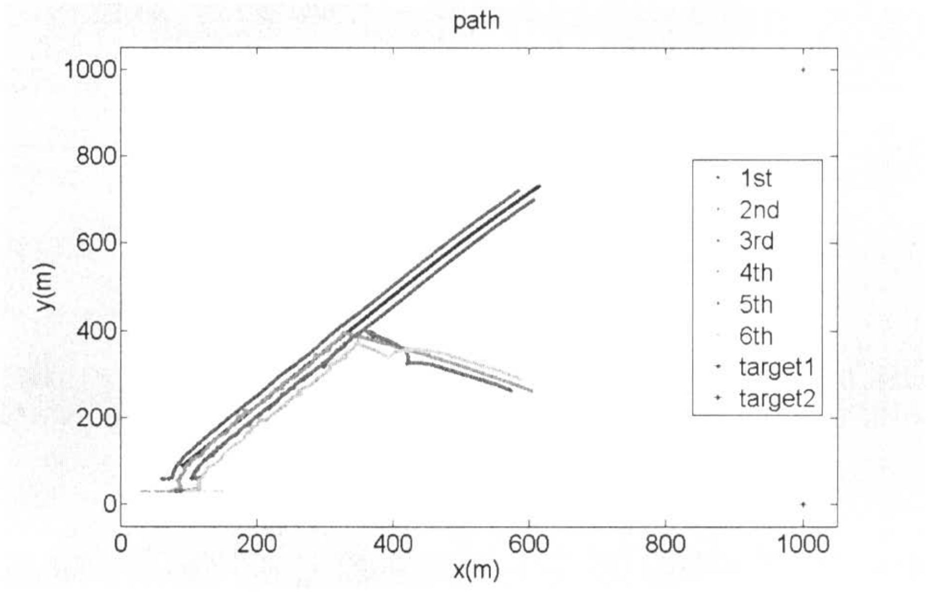


Figure 9- dynamic formation simulation

However, this dynamic formation simulation in Figure 9 is only a preliminary test for the feasibility of the concept of the subsystem concept. Therefore, there is no obstacle avoidance simulation at this point.

8.2.2 Real-Time Scheduling of Distributed RHC

All of the simulation presented in this chapter, considers the hovercraft model. Therefore, the miniature hovercraft model under study is presented.

8.2.2.1 Hovercraft Model

A complete discussion on the modeling of the RC hovercraft is presented in [37]. Since only viscous friction is considered in the nominal model, the model is similar to that presented in [22]. From the results of [37], it is straightforward to assume a linear relationship between the applied voltage and the thrust produced by the propellers. Therefore, based on Figure 5, the nominal model expressed in local coordinate is given as:

$$
\begin{aligned}
\dot{u} &= \frac{1}{M}\left(a_r V_r + a_l V_l - b_1 u\right) + v r \\
\dot{v} &= -\frac{1}{M} b_2 v - u r \\
\dot{r} &= \frac{l}{2J}(a_l V_l - a_r V_r) - \frac{1}{J} b_3 r
\end{aligned}
\tag{31}
$$

where M is the mass, J is the moment of inertia about the Z-axis, and b_1, b_2, and b_3 are the coefficients of viscous friction in X, Y, and rotational directions, respectively. Also, V_r and V_l are the applied voltages to the right and left propeller DC motors, respectively.

Another model of the RC hovercraft with more details is considered as the actual model of the system and used in simulations. This model is as follows [37]:

$$
\begin{aligned}
\dot{\omega}_{p,r} &= \frac{1}{J_a}\left(\frac{k_{e2}}{R}\left(V_r - k_{e1}\omega_{p,r}(t)\right) - k_{a3}\omega_{p,r}{}^2 - k_{a4}\omega_{p,r}(u - \frac{l}{2}r)\right) \\
\dot{\omega}_{p,l} &= \frac{1}{J_a}\left(\frac{k_{e2}}{R}\left(V_l - k_{e1}\omega_{p,l}(t)\right) - k_{a3}\omega_{p,l}{}^2 - k_{a4}\omega_{p,r}(u + \frac{l}{2}r)\right) \\
\dot{u} &= \frac{1}{M}\left(k_{a1}\omega_{p,r}{}^2 - k_{a2}\omega_{p,r}\left(u - \frac{l}{2}r\right) + k_{a1}\omega_{p,l}{}^2 - k_{a2}\omega_{p,l}\left(u + \frac{l}{2}r\right) - f_{f,u}\right) - g\sin(\theta) + vr \\
\dot{v} &= -\frac{f_{f,v}}{M} - g\sin(\phi) - ur \\
\dot{r} &= \frac{l}{2J}\left(\left(k_{a1}\omega_{p,l}{}^2 - k_{a2}\omega_{p,l}\left(u + \frac{l}{2}r\right)\right) - \left(k_{a1}\omega_{p,r}{}^2 - k_{a2}\omega_{p,r}\left(u - \frac{l}{2}r\right)\right)\right) - \frac{f_{f,r}}{J}
\end{aligned}
\tag{32}
$$

where J_a is the moment of inertia of propeller and rotating part of the DC motor, $\omega_{p,r}$ and $\omega_{p,l}$ are the angular speed of the right and left propellers, respectively. θ and ϕ are the pitch and roll angels of hovercraft indicating the ground slop. $f_{f,u}$, $f_{f,v}$, and $f_{f,r}$ are the friction functions presented in [37].
This model includes actuator dynamics, propeller dynamics, effect of hovercraft speed on the thrust produced by propeller, nonlinearities in friction applied in hovercraft, and the ground inclination. In addition, noises in measurements are applied in simulations.

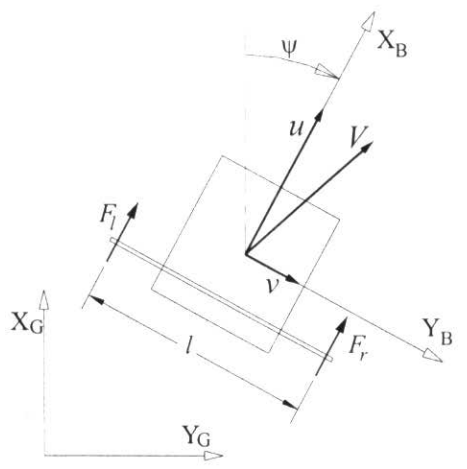


Figure 10- RC hovercraft model with produced thrusts by propellers and local (body) and global coordinate systems

8.2.2.2 Scheduling Real-Time Simulation

Real-time simulations for scheduling of multiple hovercrafts on multiple processors are presented. These simulations are based on the proposed scheduling method presented in [2] and explained briefly in chapter 2. The problem is to determine the execution horizon and communication period of every vehicle, dynamically based on the overall performance of the system. The problem is illustrated for the case of three processors in Figure 4.

In this section, the vehicles are considered in the leader-follower case. The leader produces the trajectory while the followers maintain a formation with respect to each others and the leader. Two cases are studied:

Case1: two processors, each one having two vehicles

Case2: four processors, each one having a single vehicle

8.2.2.2.1 Scheduling Real-Time Simulation (Case I)

Formation of four vehicles is considered in this example. Two computers (processors) are used and each one has two hovercrafts (Figure 11). Different communication delays and communication bandwidths are studied in this example.

Figure 11- Schematic diagram of case 1; two computers and four vehicles

The nominal model of hovercrafts is presented in (31). All of them have similar nominal model, expressed by the following parameters:

(Parameters of the nominal model for all hovercrafts)

The actual model of these hovercrafts is expressed by (32). (add g to equations)

It is supposed that hovercrafts 1, 2, and 4 have same levels of uncertainties and equal to b=0.1 (See **Proposition 1** for definition of *b*), and hovercraft 3 has higher uncertainty b=3.0. Note that all units in this example are in SI Units.

The controller parameters are similar for all of the systems and are as follows:
(controller parameters in a table)

The cost function associated to each hovercraft can be expressed as:
Hovercraft 1 (leader)
(cost function)

Hovercrafts 2, 3, 4 (followers)
(cost function)

The communication delay between two computers is around 0.2 s. In the following figures, the results for the case of trajectory following is presented.

Figure 12- Trajectory, followed by the hovercrafts

Figure 13- Snapshots in different times to show the formation

Figure 14- Computation scheduling results for both CPUs (execution horizon vs. time)

Figure 15- Communication scheduling results (communication period vs. time)

In order to emphasize the effectiveness of the proposed dynamic scheduling algorithm, the similar situation is simulated with a static scheduling algorithm and the result of trajectory following is presented in the following figures.

Figure 16- Trajectory, followed by the hovercrafts

Figure 17- Snapshots in different times to show the formation

The differences in comparing figures (13) and (14) to figures (17) and (18), is only because of the scheduling method.

8.2.2.2.2 Scheduling Real-Time Simulation (Case II)

Formation of four vehicles is studied on the cluster of four computers. In this example, distributed RHC and Communication scheduling is focused. Since obstacle avoidance was simulated in section (8.2.1), it is not considered in this example, and the case of dynamic formation is added.

The hovercrafts used in this example were similar to the previous case. The distances of vehicles are changing while following their trajectory. The cost functions and controller parameters were similar to the previous case. The results are presented in the following figures.

Figure 18- Trajectories, followed by the hovercrafts

Figure 19- Snapshots in different times to show the actual formation and the desired formation

8.3 Experimental Verification

In this chapter, the result of applying RHC to miniature helicopter and hovercraft is investigated. Dynamic scheduling algorithms explained in Chapter 2 for multiple systems on distributed computers were verified using real-time simulations in chapter 3. In this chapter, these methods are applied experimentally to multiple hovercrafts to verify the applicability of the method. In addition, the RHC method was applied for trajectory following of a miniature helicopter, and is presented here.

In the following chapters, first, the apparatus is described and every component is explained; after that, the results are presented for different cases.

8.3.1 Description of the Apparatus

Figure 21 shows a schematic diagram of the setup. The setup consists of the following parts:

4- A set of cameras (C1 to C9) used to capture images and to forward to the Vision computers (V1 to V9). Each image consists of some colored objects and the computers calculate the centroid of each object (Figure 21). Each camera is connected to one computer in order to decrease the image processing time.

5- All of the centeroids calculated by the vision computers are transferred to the server via a Gigabyte Ethernet Switch. The server combines the centroid information and sends them to the main controller computers (M1 to M4) via another Gigabyte Ethernet Switch.

6- M1 and M2 receive the data from sensors. One sensor is the vision system and the data is sent from the server, and the other sensor is a wireless orientation sensor (Wireless InertiaCube3 [24]) and calculate the input. The input signals are transferred to the apparatus via data acquisition board and wireless transmitter[35] (T1 and T2). In this setup each computer is connected to one apparatuses.

[35] The wireless transmitter is "HiTec Laser 6 Channel 75MHz RC Controller Pkg" and is explained in [28] and [31].

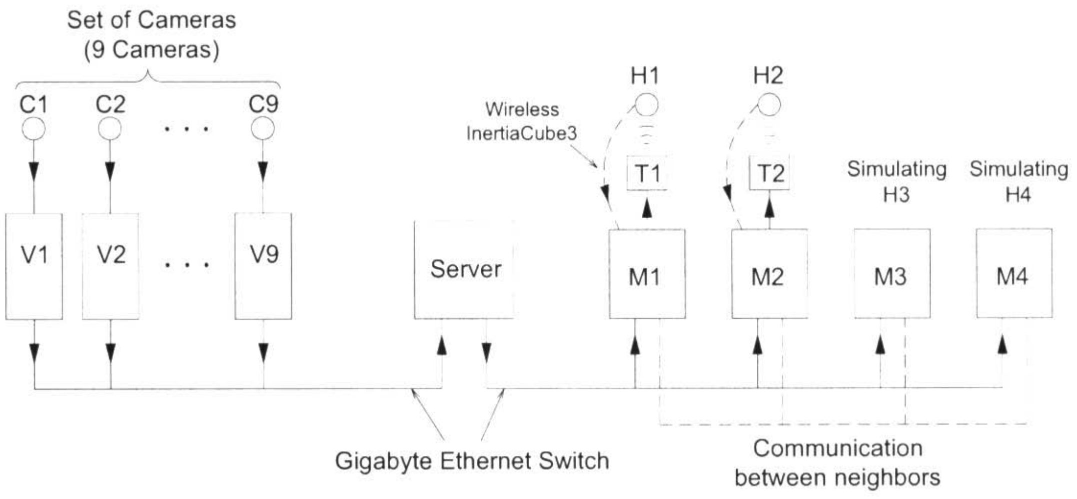


Figure 20- Schematic layout of experimental setup

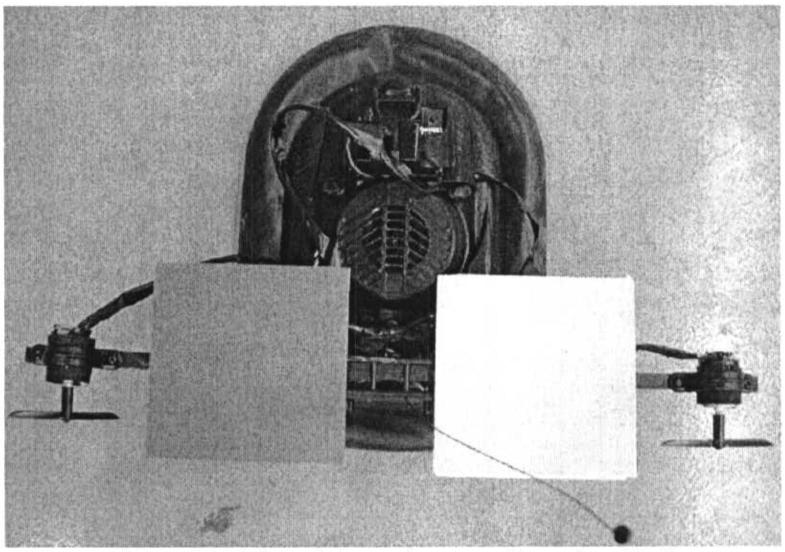


Figure 21– The miniature hovercraft version H1.0

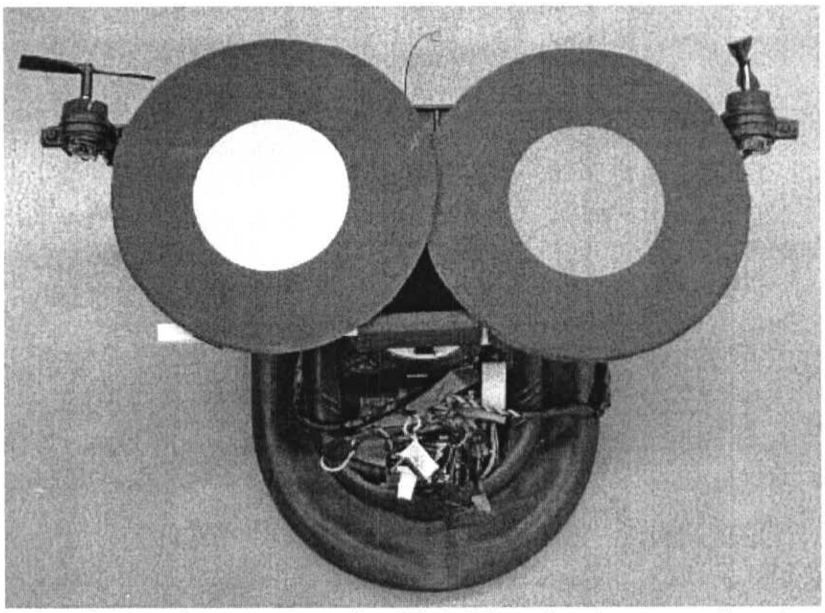


Figure 22– The miniature hovercraft version H1.1. The colored squares of version H1.0 were changed to colored circles as shown and had better measurement results.

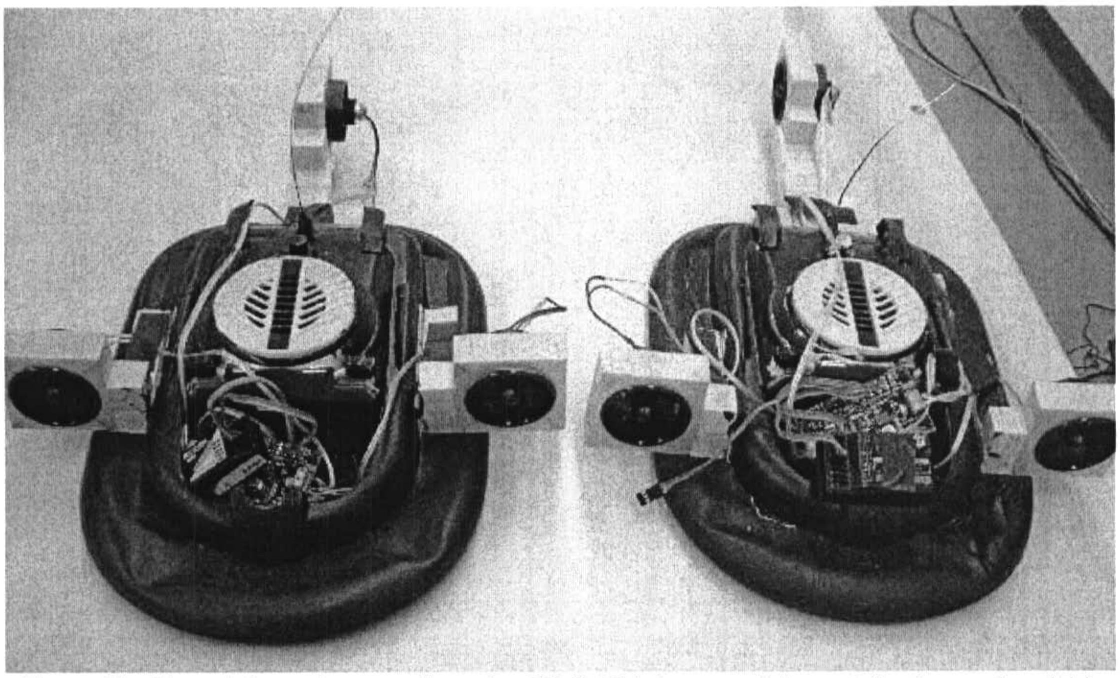


Figure 23– The miniature hovercraft version H2.0. This hovercraft is used for the results of this report. It has more powerful fans and its model is more similar to 3DOF miniature helicopter.

The apparatus used in this investigation is the miniature hovercraft shown in Figure 23. The previous versions of hovercrafts are also shown in figures 21 and 22. This hovercraft (Figure 23) is a modified version of the R/C hovercraft [32]. The thrusters are ducted fans [39] and they are more powerful than the previous fans.

To investigate the performance of RHC approach with trajectory tracking of helicopters, the miniature helicopter as shown in Figure 24, is used.

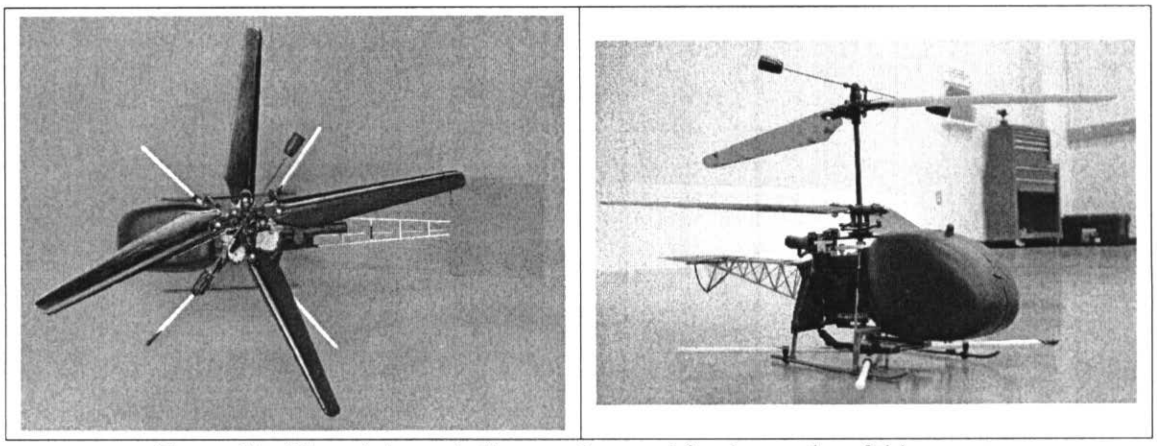


Figure 24– The miniature helicopter that used for the results of this report.

For controlling the apparatus, a wireless transmitter is attached to the computer using a data acquisition board and the receiver is attached to the apparatus. In order to connect the transmitter to the data acquisition board some modifications are done in the transmitter. The data acquisition board, transmitter and all of the modifications are described in [28]. Two batteries are used as follows:

3- One 9.6 V battery used for the main fan which cause flouting of hovercraft. This fan pushes air under the hovercraft and causes it to move on a film of air, therefore the friction of hovercraft in all directions (forward, side, and angular) is very small.
4- One 7.4 V battery used for all ducted fans. This battery is connected to the receiver and all fans are connected to the receiver.

As mentioned above, in order to measure orientation and angular velocity of miniature hovercraft, a wireless sensor is used. This sensor is a Wireless InertiaCube3 from InterSence [24]. This sensor is described and its benefits are explained in [3]. The position in X-Y coordinate is obtained using Vision System which is developed in the CIS Lab and is illustrated in the following. In addition, for performing these experiments a new surface without slope is prepared recently in CIS Lab.

8.3.1.1 Vision Array

In order to have the position (X-Y) of the hovercraft a vision array is built at a height of 3.5 meters in the test area of the CIS Lab. It produces a test area in the shape of a square with dimensions of 20 by 20 feet. 9 webcams[36] are connected to 9 computers and they send the data to a main computer which acts as a server. See Figure 21, and Figure 36 to

[36] Installing more webcams is also possible but with four webcams the working area is covered with a reasonable resolution.

Figure 38. Figure 36 shows an overview of the vision array. Figure 37 is a schematic drawing showing a connection point. Figure 38 shows a connection point and a webcam.

Each computer with a webcam connected to it can get the picture and process the image within less than 0.05 of a second. Therefore, the data from the whole vision system is updated in less than 0.05 of a second.

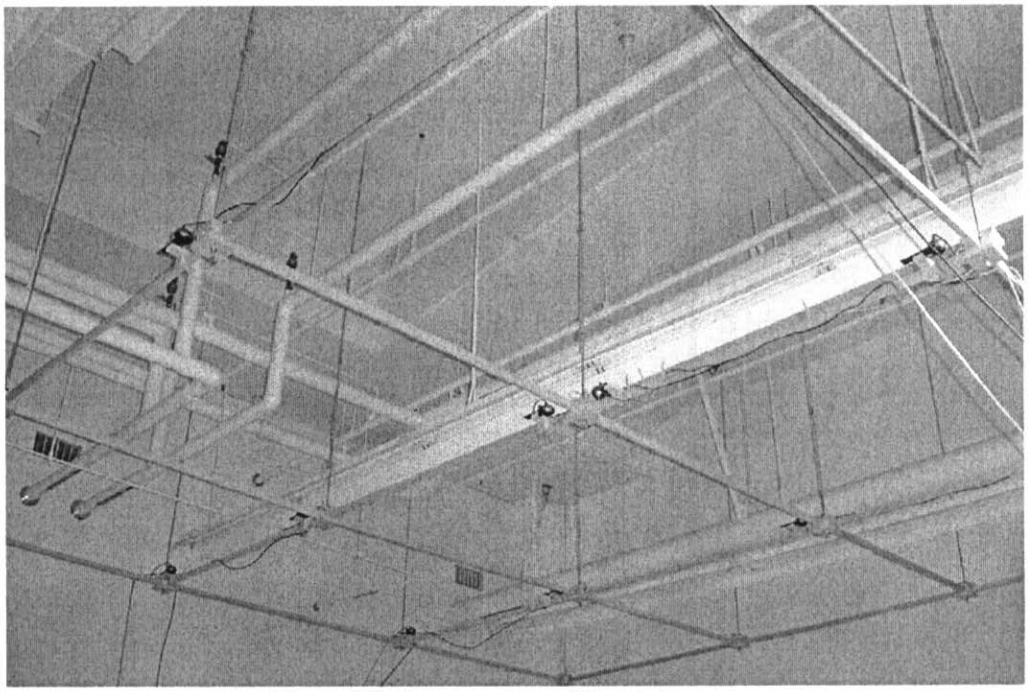


Figure 25- An overview of the vision array

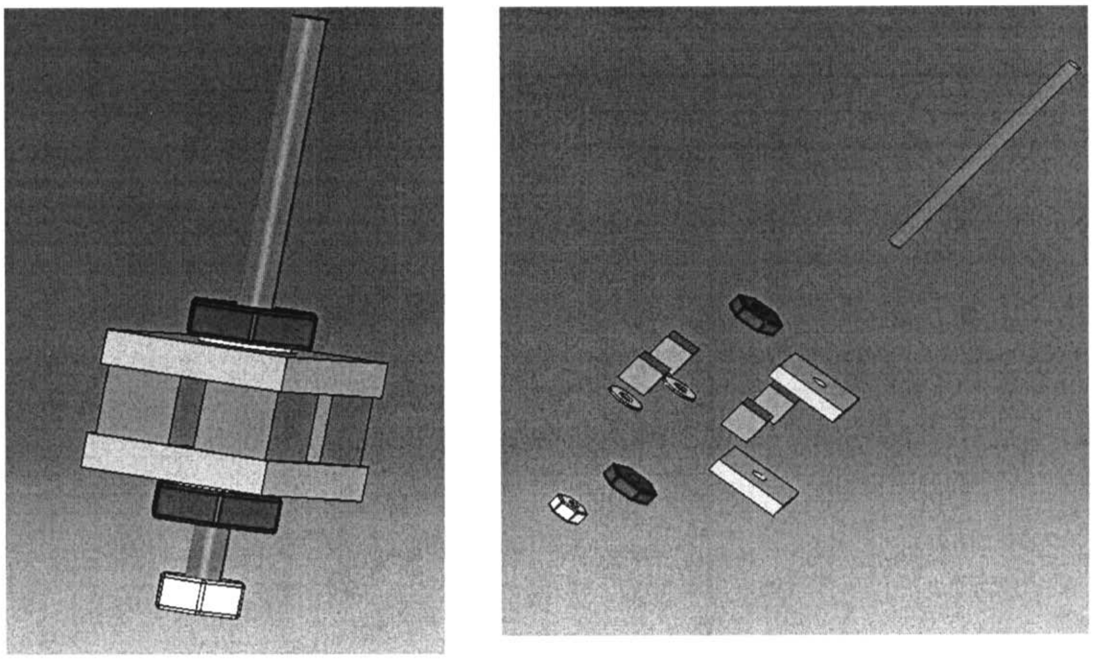


Figure 26- Schematic drawing of a connection point

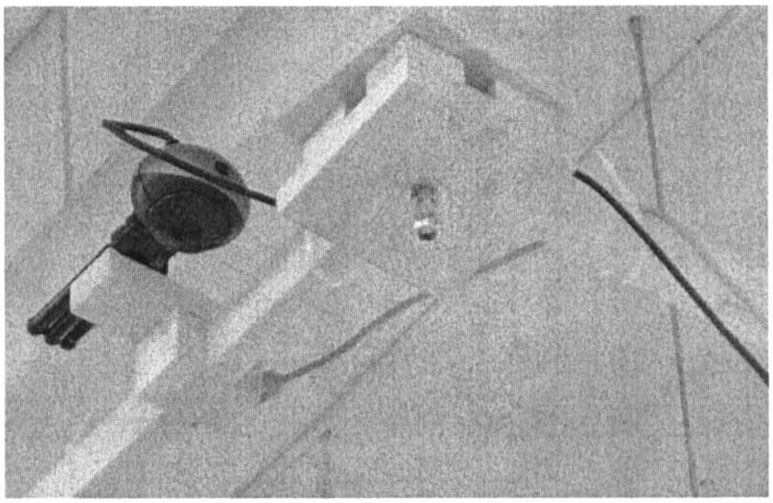


Figure 27- A connection point with a webcam

8.3.2 Formation of Four Hovercrafts

The experimental setup is shown in Figure 21. As illustrated in that figure, two miniature hovercrafts are used and two hovercrafts are being simulated.
Nominal models of all vehicles are expressed with the following equations:

(Equations of motion for hovercraft H2.0)

where … (explanation of parameters).

Using a similar identification algorithm explained in [3], the parameters of both hovercrafts are identified as follows:

(Parameters of hovercraft 1)

(Parameters of hovercraft 2)

For hovercrafts 3 and 4, the parameters are considered to be similar to system 1.

The RHC controller used in this experiment was adapted from [6] and RHCOOL [3] is used for implementation.
Parameters of RHC controller for all of the subsystems are as follows:

Hovercraft 1: (change the table)

Nx = 6	Number of states $(u, v, r, \psi, x_c, y_c)$
Nz = 2	Number of flat outputs (ψ, u)
Nu = 2	Number of inputs (u_1, u_2)
Ny = 2	Number of outputs
Nd = 2	Maximum number of derivatives from flat outputs
Ns = 4	Number of Spline control points
Ni = 101	Number of Spline interpolation points
T = 4.0 (sec)	Prediction Horizon
Nc = Ns	Number of varying control points
Imethod = 1	It means that the cubic spline is used for interpolation
Q	
R	

Hovercraft 2:
(Add the table)

Hovercrafts 3 and 4: similar to hovercraft 1

In Figure 21 the computers used for controlling the hovercrafts (M1 to M4) are all Pentium 4, with one CPU of 2.8 GHz speed.

This example is a leader follower with a fixed formation. Hovercraft 1 is the leader and the others are followers. Dynamic scheduler defines the communication rate for each vehicle. In the following figures, trajectory, inputs, and communication period of all systems are presented.

Figure 28- Followed trajectories of all vehicles and the reference trajectory.

Figure 29- Communication periods of all vehicles versus time.

Figure 30- Inputs applied to the leader (hovercraft 1)

Figure 31- Inputs applied to hovercraft 2

Figure 32- Change of scheduling parameters over time. (Online estimation of scheduling parameters)

8.3.3 Trajectory Tracking of Miniature Helicopter Using RHC

The miniature helicopter shown in Figure 24 is used in this test. The vision array is used to obtain necessary information. The experimental setup in this test is presented schematically in Figure 33.

Figure 33- experimental setup for miniature helicopter

The model used in the controller is as follows:

(Nominal model for miniature helicopter)

The parameters of this helicopter were identified using a similar approach explained in [3] and are as follows:

(Parameters of miniature helicopter)

As mentioned before, the RHC approach is used and RHCOOL [3] version 4.0 is used for real-time implementation. The parameters of controller are as follows:

(Controller parameters)

The following figure, presents the result.

Figure 34- Trajectory of the miniature helicopter and the reference trajectory

8.4 Conclusions

In this report, a systematic approach is developed for dynamic scheduling of computation and communication to determine the execution horizon and communication rate for multiple uncertain RHC systems on distributed processors. It was shown that using the scheduling methods combined with analytical bounds on the prediction error, the problem of scheduling multiple uncertain plants can be cast into an appropriate constrained optimization problem. The constraints guarantee that the processes will be schedulable, both computation and communication, and the optimization provides performance robustness to uncertainty. The proposed method was first applied via C++ simulations to a set of hovercrafts in formation (leader-follower) on distributed digital processors and it demonstrated the validity of the approach. Performance and applicability of the method was also demonstrated by applying it experimentally, to some sets of under actuated miniature hovercrafts in a hard real time environment for a leader-follower example. Both of the simulation and experiment were done by developing an RHC Object Oriented Library (RHCOOL) in C++. This report was explaining Task #12, Task #13, and Task #14 of the first proposal.

8.5 References

[1] A. Azimi, B. W. Gordon, C. A. Rabbath, "Dynamic Scheduling of Receding Horizon Controllers with Application to Multiple Unmanned Hovercraft Systems", *Accepted on American Control Conference (ACC)*, 2007.

[2] A. Azimi, B. W. Gordon, "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques - Real-time Dynamic Scheduling Algorithm for Multiple RHC Running on Distributed Digital Processors", *DRDC Technical Report*, Feb. 2007.

[3] A. Azimi, B. Gholami, B. W. Gordon, "Synthesis and Implementation of Single and Multi-vehicle Systems Guidance Based on Nonlinear Control and Optimization Techniques - Real-time Scheduling of Multiple Uncertain Receding Horizon Control Systems", *DRDC Technical Report*, March 2006.

[4] Y. Zhao, B. W. Gordon, "Distributed Receding Horizon Control of Multi-Vehicle Systems", Submitted to the 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2007).

[5] M. B. Milam, R. Franz, J. E. Hauser, R. M. Murray, "Receding horizon control of a vectored thrust flight experiment", *submitted to IEE Proceedings on Control Theory and Applications*.

[6] D. Henriksson, J. Akesson, 2004, "Flexible implementation of Model Predictive Control using sub-optimal solutions", *Technical Report, Department of Automatic Control, Lund Institute of Technology*, Lund, Sweden.

[7] H. Kopetz, "Real-time systems: Design principles for distributed embedded applications", Chapters 9 and 11, *Springer*, 1997.

[8] C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of ACM*, 20 (1), 1973.

[9] D. Henriksson, "Flexible scheduling methods and tools for real-time control systems", *Licentiate Thesis*, December 2003. Department of Automatic Control, Lund Institute of Technology, Sweden.

[10] Chen, H., Allgower, F., 1998, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability", *Automatica*, 34, pp.1205-1217.

[11] Jadbabaie, A., 2000, "Receding horizon control of nonlinear systems: A control Lyapunov function approach", *Ph.D Thesis, California Institute of Technology*, Pasadena, CA.

[12] B. Gholami, B. W. Gordon, C. A. Rabbath, "Real-time scheduling of multiple uncertain receding horizon control systems", *Technical Report 07-05*, Control and Information Systems Laboratory, Concordia University, Montreal, 2005.

[13] H. Khalil, *Nonlinear Systems*, Prentice Hall, 2002.

[14] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems", *PhD Thesis, California Institute of Technology,* Pasadena, CA, 2003.

[15] B. J. Young, R. W. Beard, J. M. Kelsey, "A control scheme for improving multi-vehicle formation maneuvers", *American Control Conference*, Arlington, VA, pp. 704-709, 2001.

[16] Desai, J. P., Ostrowski, J., Kumar, V., 1998, "Controlling formations of multiple mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp. 2864-2869.

[17] Pomet, J.-B., Thuliot, B., Bastin, G., Campion, G., 1992, "A hybrid strategy for the feedback stabilization of nonholonomic mobile robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, pp. 129 – 134.

[18] Feddema, J. T., Lewis, C., Schoenwald, D. A., 2002, "Decentralized control of cooperative robotic vehicles: theory and application", *IEEE Transaction on Robotics and Automation*, 18 (5), pp. 852-864.

[19] J.-P. Aubin, A. Cellina, *Differential inclusions: set-valued maps and viability theory*, Springer-Verlag, 1984.

[20] A. P. Aguiar, L. Cremean, and J. P. Hespanha, "Position Tracking for a Nonlinear Underactuated Hovercraft: Controller Design and Experimental Results", CDC, 2003.

[21] Wireless InertiaCube3 Supplemental Manual, Doc. No. 072-00096-0E05, InterSense, Inc, 2005.

[22] InertiaCube3 Manual, Doc. No. 072-00094-0D05, InterSense, Inc, 2005.

[23] V. Gavrilets, B. Mettler, E. Feron, "Dynamic Model for a Miniature Aerobatic Helicopter", Technical Report, MIT.

[24] X. Ren, B. W. Gordon, "Helicopter control system and 3D motion tracking", Technical report, CIS Lab, 12/2003

[25] C. W. Mercer, "An Introduction to Real-Time Operating Systems: Scheduling Theory", School of Computer Science, Carnegie Mellon University, 1992.

[26] http://www.superdroidrobots.com/shop/item.asp?itemid=601

[27] http://www.amazon.com/exec/obidos/tg/detail/-/B0006TQ61M/qid=1143092249/br=1-14/ref=br_lf_t_14//104-4082232-4929547?v=glance&s=imaginarium&n=13685221

[28] http://www.rcuniverse.com/product_guide/servoprofile.cfm?servo_id=79

[29] http://www.intersense.com/support/faqs/ic2.htm

[30] http://www.sce.carleton.ca/courses/sysc-4102/hard-real-time.pdf

[31] http://www.nag.co.uk/numeric/cl/nagdoc_cl08/pdf/E04/e04ucc.pdf

[32] http://www.nag.co.uk/numeric/cl/nagdoc_cl08/pdf/E04/e04xzc.pdf

[33] http://www.sbsi-sol-optimize.com/manuals/SNOPT%20Manual.pdf

[34] http://www.cs.bu.edu/~best/crs/cs835/S96/lectures/scheduling.html

[35] G. C. Buttazzo, "Rate Monotonic vs. EDF: Judgment Day", *Real-Time Systems*, 29, 5–26, 2005.

[36] http://feanor.sssup.it/~giorgio/slides/rts/HB.pdf

[37] A. Azimi, B. W. Gordon, "Modeling of RC Hovercraft", Technical Report, CIS Lab, Concordia University, July 2006.

[38] A. Azimi, B. W. Gordon, "discussion on Gronwall-Belmann Lemma for van Der Pol Oscillator and RC hovercraft problem", Technical Report, CIS Lab, Concordia University, September 2006.

[39] Ducted fan

[40] Feng-Li Lian and Richard Murray, "Real-Time Trajectory Generation for the Cooperative Path Planning of Multi-Vehicle Systems", California Institute of

Technology

[41] David H. Shim, H. Jin Kim, Shankar Sastry, "Decentralized Reflective Model Predictive Control of Multiple Flying Robots in Dynamic Environment", Department of Electrical Engineering & Computer Sciences, University of California at Berkeley

[42] Yang Bo, Wang Qinghua, "Agent Brigade in Dynamic Formation of Robotic Soccer", Proceeding of the 3rd World Congress on Intelligent Control and Automation, pp.174-178, June, 2000

8.6 Appendix H: Explanation of Attachments

In this chapter, the experimental results and the codes written for the materials of this report, which are included in the attached CD, is described. The data is classified based on the different chapters of this report.

Chapter 3

All of the data used for chapter 3 of the report are included in this folder. It includes different subfolders based on the report and the contents of each folder are described bellow.

Section 3.2

All of the supporting data for section 3.2 of the report is included in this folder. It has the following subfolders:

Chapter 6

In this folder, some of the references are included for more convenience of the reader.